AFRL-IF-RS-TR-2007-19
Final Technical Report
January 2007

DARPA AGENT MARKUP LANGUAGE (DAML) TOOLS FOR SUPPORTING INTELLIGENT ANNOTATION, SHARING AND RETRIEVAL

University of Maryland, Baltimore

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K528

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Rome Research Site Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-IF-RS-TR-2007-19 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
RANDALL J. MCINTYRE Work Unit Manager	JAMES W. CUSACK, Chief Information Systems Division Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JAN 2007	Final	Jun 00 - Apr 06

4. TITLE AND SUBTITLE

DARPA AGENT MARKUP LANGUAGE (DAML) TOOLS FOR SUPPORTING INTELLIGENT ANNOTATION, SHARING AND RETRIEVAL

5a. CONTRACT NUMBER

5b. GRANT NUMBER
F30602-00-2-0591

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)

Timothy Finin, James Mayfield and Benjamin Grosof

5d. PROJECT NUMBER
DAML

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
12

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Maryland, Baltimore
1000 Hilltop Circle
Baltimore MD 21250

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Dr.
Arlington VA 22203-1714

AFRL/IFSA
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2007-19

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 07-025

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The UMBC lead DAML project began with the goal to design and prototype critical software components enabling developers to create intelligent software agents capable of understanding and processing information and knowledge encoded in DAML and other semantically rich markup languages. The project was led by Tim Finin (UMBC) who served as the PI, CO-PI Jim Mayfield (JHU) and CO-PI Benjamin Grosof (MIT). The work had three main research thrusts: exploring and evaluating how semantic web technology can be integrated into and used by agent based systems, developing techniques for building information retrieval systems using semantic web data and knowledge and developing and evaluating better rule based systems for the semantic web. This final report describes the most significant results and provides a complete list of papers that provide more information of the research carried out and detailed results.

15. SUBJECT TERMS
DARPA Agent Markup Language (DAML), software agent, semantic web technology, agent based systems, information retrieval, rule based systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Randall J. McIntyre
U	U	U	UL	113	19b. TELEPHONE NUMBER (*Include area code*)

Table of Contents

OVERVIEW **1**

THE ITTALKS SYSTEM **2**

ITTALKS: A CASE STUDY IN THE SEMANTIC WEB AND DAML+OIL 3

THE SWOOGLE SEMANTIC WEB SEARCH ENGINE **13**

SEARCH ENGINES FOR SEMANTIC WEB KNOWLEDGE 13
CHARACTERIZING THE SEMANTIC WEB ON THE WEB 32

SWEETJESS AND SWEETRULES **48**

SWEETJESS: INFERENCING IN SITUATED COURTEOUS RULEML VIA TRANSLATION TO AND FROM JESS RULES 50
SWEETRULES AND SWEETDEAL 67

THE TAGA AGENT FRAMEWORK **82**

USING SEMANTIC WEB TECHNOLOGY IN MULTI-AGENT SYSTEMS: A CASE STUDY IN THE TAGA TRADING AGENT ENVIRONMENT 83
F-OWL: AN INFERENCE ENGINE FOR THE SEMANTIC WEB 90

PAPERS PRODUCED **102**

LIST OF ACRONYMS **109**

Overview

The UMBC lead DAML project began with the goal to design and prototype critical software components enabling developers to create intelligent software agents capable of understanding and processing information and knowledge encoded in DAML and other semantically rich markup languages. The project was led by Tim Finin (UMBC) who served as the PI, CO-PI Jim Mayfield (JHU) and CO-PI Benjamin Grosof (MIT). The work had three main research thrusts: exploring and evaluating how semantic web technology can be integrated into and used by agent based systems, developing techniques for building information retrieval systems using semantic web data and knowledge and developing and evaluating better rule based systems for the semantic web. This final report describes the most significant results and provides a complete list of papers that provide more information of the research carried out and detailed results.

Given the length of the project and the fact that it involved a team of three organizations working sometimes independently and sometimes together, it is difficult to provide a single set of results. Rather, we list one or two major results in each of the three major tasks. To provide detail, we include a number of comprehensive papers as a part of this report. A complete list of papers describing the research and results that were supported by or partially supported by this project is given at the end of this report. Most of these are available on the web and can be obtained by following the links provided. We also list some of the software packages and systems developed by the project.

The ITTalks System

ITtalks is a Web portal that offers access to information about talks, seminars, and colloquia related to information technology. ITtalks users can view such information as location, speaker, hosting organization, and talk topic. ITtalks also lets agents retrieve and manipulate information stored in the ITtalks knowledge base. ITtalks used DAML+OIL for knowledge base representation, reasoning, and agent communication. We used DAML+OIL to mark up stored information to provide additional reasoning capabilities. Additionally, ITtalks uses DAML+OIL for all communication, including simple messages and queries, using the ITtalks-defined ontology.

The ITtalks framework was used to support a range of research including the development of tools like SweetJess and DAMLator, prototype agent applications, interoperability experiments and ontologies.

ITtalks: A Case Study in the Semantic Web and DAML+OIL

For details, we include a paper that appeared in the January issue of IEEE Intelligent Systems.

ITtalks: A Case Study in the Semantic Web and DAML+OIL

R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff, Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, and Sovrin Tolia,
University of Maryland, Baltimore County

Semantic Web markup languages will improve the automated gathering and processing of information and help integrate multiagent systems with the existing information infrastructure. In this article, the authors describe their ITtalks system and discuss how Semantic Web concepts and DAML+OIL extend its ability to provide an intelligent online service.

Semantic Web developers seek to make the Web more machine-readable, to let intelligent agents retrieve and manipulate pertinent information. Achieving this will require seamless agent integration with the Web and taking full advantage of the existing infrastructure (such as message sending, security, authentication, directory services, and application service frameworks). We believe that the Semantic Web markup language DAML+OIL (DARPA Agent Markup Language plus Ontology Inference Layer) will be central to this goal's realization. The DAML Program aims to develop a Semantic Web markup language that provides sufficient rules for ontology development and that supports intelligent agents and other applications.[1,2] (For more information, see the "DAML+OIL" sidebar.)

As part of the University of Maryland, Baltimore County's role in the DAML Program, we constructed a fielded application, ITtalks, which facilitates user and agent interaction for locating talks on information technology. ITtalks also lets developers design and create intelligent software agents that can understand and process information encoded in DAML+OIL or other semantically rich markup languages. Primarily, we focused on developing the support and infrastructure for integrating intelligent agents into the Web.

ITtalks features

ITtalks is a Web portal that offers access to information about talks, seminars, and colloquia related to information technology. ITtalks users can view such information as location, speaker, hosting organization, and talk topic. ITtalks also lets agents retrieve and manipulate information stored in the ITtalks knowledge base.

ITtalks uses DAML+OIL for knowledge base representation, reasoning, and agent communication (see Figure 1). We used DAML+OIL to mark up stored information to provide additional reasoning capabilities. Additionally, ITtalks uses DAML+OIL for all communication, including simple messages and queries, using the ITtalks-defined ontology.

Personalized accounts

You can access ITtalks anonymously or through a personalized account. You register by entering information manually through Web forms or by providing the location (URL) of a DAML+OIL personal profile, which could include your location, interests, contact details, and schedule. Your schedule could be rudimentary (a list of available time periods for given days) or significantly more detailed. ITtalks uses this information to personalize the site, displaying only talks that match your interests, schedule, or both.

Because DAML+OIL is not yet in widespread use, ITtalks provides a tool for creating a DAML+OIL personal profile. The tool constructs a profile containing only items that the ITtalks system uses. However, we believe that the profile will ultimately provide a unique and universal point for obtaining personal user information for all information needs, not just for ITtalks, and could then include any information you wish to share.

Customized domains

To help create a universal resource for the international IT research community, we organized ITtalks around domains, which typically represent event-hosting organizations such as universities, research laboratories, or professional groups. A separate Web site exists per domain, each independently maintained by a moderator who can define the domain's scope

1094-7167/02/$17.00 © 2002 IEEE

[Stone, 2000] Peter Stone and Amy Greenwald: The First International Trading Agent Competition: Autonomous Bidding Agents, Electronic Commerce Research Journal pp1-36, 2000.

[Wellman, 2002] Michael P. Wellman, Amy Greenwald, Peter Stone, and Peter R. Wurman: The 2001 Trading Agent Competition, Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2002), pp935-941, Edmonton, August 2002.

[Willmott, 2001] Willmott, S., Dale, J., Burg, B., Charlton, P. and O'Brien, P., Agentcities: A Worldwide Open Agent Network. In: AgentLink News, Issue 8, November 2001.

[Yang, 2000] Guizhen Yang and Michael Kifer. FLORA: Implementing an efficient DOOD system using a tabling logic engine. Proceedings of Computational Logic --- CL-2000, number 1861 in LNAI, pp 1078--1093. Springer, July 2000.

[Zou, 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan, TAGA: Trading Agent Competition in Agentcities, Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, Monday, 11 August, 2003, Acuulco MX.

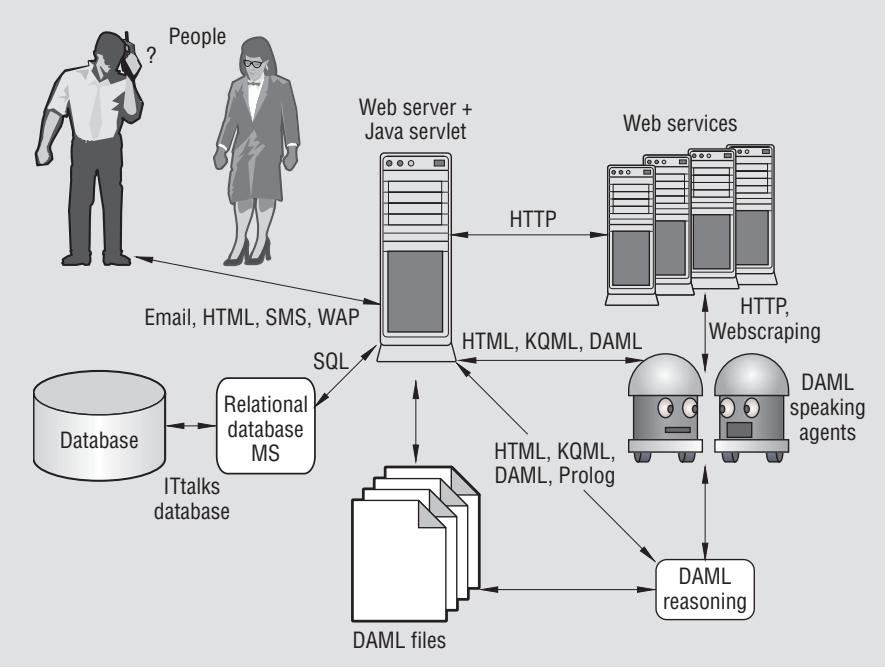

Figure 1. The architecture for ITtalks is built around a Web server backed by a relational database. ITtalks has interfaces for human users, software agents, and Web services.

```
<rdf:RDF>
 <rdf:Description about="&itlink;742">
  <Talk rdf:parseType="Resource"
   <Title>Invisible Web</Title>
   <Abstract>
    Accessing information anytime, anywhere has...
   </Abstract>
   <BeginTime>
    <time:Year>2002</time:Year>
    <time:Month>01</time:Month>
    <time:Day>17</time:Day>
    <time:Hour>12</time:Hour>
    <time:Minute>00</time:Minute>
    <time:Second>00</time:Second>
   </BeginTime>
   <EndTime>
    <time:Year>2002</time:Year>
    <time:Month>01</time:Month>
    <time:Day>17</time:Day>
    <time:Hour>13</time:Hour>
    <time:Minute>00</time:Minute>
    <time:Second>00</time:Second>
   </EndTime>
   <Location rdf:parseType="Resource">
    <Institution>UMBC</Institution>
    <Building>Math-Physics</Building>
    <Room>401</Room>
    <Street1>1000 Hilltop Circle</Street1>
    <City>Baltimore</City>
    <State>MD</State>
    <Zip>21250</Zip>
    <Country>USA</Country>
   </Location>
  …
    <Talk>
  </rdf:Description>
 </rdf:RDF>
```

Figure 2. An example of DAML+OIL-encoded knowledge.

talks and the users' interests throughout the system. The DAML+OIL ontologies let users add assertions in DAML+OIL to further characterize their interests. The topic ontologies support an automated talk classification, for which we obtained an ACM CCS training collection and are generating an Open Directory training collection to develop the necessary components. We are also developing a semiautomated component that can map topics in one ontology to topics in another, by using user-specific mapping *believes* and by exploiting the fact that each ontology's nodes have an associated text collection.

Data entry support

Although we have simplified data entry—by supporting automatic form completion using information from the knowledge base and the user's DAML+OIL profile—it still needs improvement. Therefore, we, along with the Lockheed Martin research group, are developing a focused Web spider to collect talk announcements from open sources on the Web. This spider will identify and automatically add key information to the ITtalks knowledge base using Lockheed Martin's AeroText information extraction system.

Architecture

ITtalks uses a relational database combined with a Web server to provide user access to the system. ITtalks also has an interface for agent-based communication.

Database

The ITtalks system uses the MySQL relational database software. We store the contents of the ITtalks knowledge base in a database whose schema is closely mapped to our ontologies, describing events, people, topics, and locations. We chose MySQL because of its known reliability and because we needed software with a license that lets us offer the ITtalks package to academic and commercial institutions. We are considering replacing MySQL with a native XML database software such as dbXML.

Web server

We used a combination of Apache and Tomcat as the Web portal servers. This lets us present IT talk descriptions to the user through Java servlets and JavaServer Pages files, which dynamically generate requested information in DAML+OIL, XML, HTML, RSS, and WML. ITtalks can also deliver information viewable on either a standard, computer-based phone or a WAP-enabled cellular phone.

A typical scenario

To better portray user interaction with ITtalks, let's look at a simple, typical interaction. In this scenario, Jim learns from his colleagues about the ITtalks Web portal as a source of IT-related events in his area; Jim is affiliated with Stanford University.

Jim directs his browser to the www.ittalks.org main page. Seeing a link to http://umbc.ittalks.org, he selects it and is presented with a new page listing upcoming talks scheduled at UMBC, Johns Hopkins University, and other locations within a 15-mile radius (the default distance for the UMBC domain).

Jim browses the Web site, viewing announcements matching his interests and preferred locations (as provided in his explicit search queries). He requests to view the talk information in WML. Finding a talk of potential interest to a colleague, Jim takes advantage of the invitation feature, which lets him send an invitational email to any of his friends for any

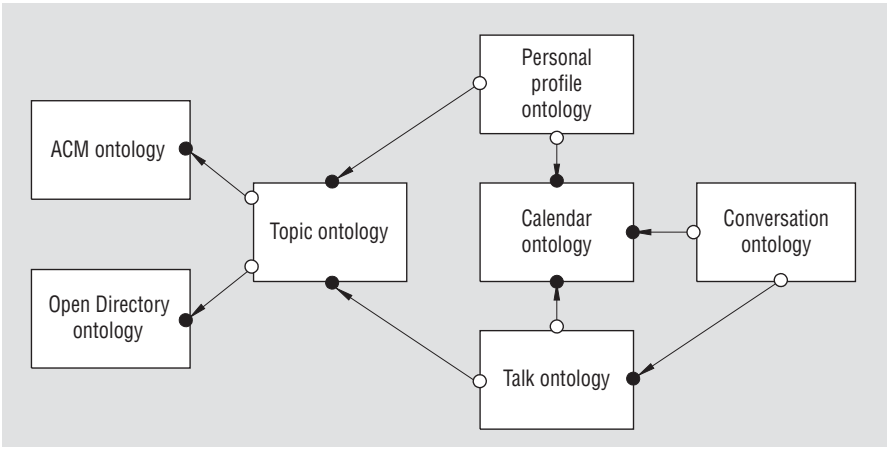

Figure 3. The relationships among the various ontologies that ITtalks uses.

of the listed talks. Finally, using the Personalize link on the bottom of the page, Jim creates his own http://ittalks.org main page, by providing the URL of his DAML+OIL-encoded profile. This customized page, listing talks based on his preferences, will be Jim's entrance to the ITtalks site whenever he returns.

Agents

We'd like ITtalks to demonstrate, among other things, how DAML+OIL facilitates integration of service agents with online information resources. To this end, and to extend ITtalks' capabilities, we defined several agents that support ITtalks' operation. You can view some as supporting services (such as external information services); we assume that others will exist in the general environment in the future.

ITtalks

The ITtalks agent is a front end for the ITtalks system. The agent interacts with ITtalks through the Web-based interface used by humans but communicates via an ACL (Agent Communication Language) with other agents on the Web. The agent acts primarily as a gateway for agent access and does not support any advanced functionality.

User

Agent research has long aimed to represent human users online through agents that can service queries and filter information for them. Although ITtalks does not require such agents to exist, we recognize the power that such agents add. Therefore, ITtalks supports interaction with user agents as well as their human counterparts. The user agent that we developed understands DAML+OIL, supports sophisticated reasoning, and communicates via a standard ACL. It reasons by means of XSB, a logic programming and deductive database system for Unix and Windows developed at the State University of New York, Stony Brook.

Calendar

Although a user agent might contain the necessary knowledge about its user's schedule, we believe that it will benefit from assigning the calendar-based facts and preferences to a separate calendar agent. This lets the user agent use the same protocol to consult the user calendar and other groups' or users' calendar agents. The calendar agent can only represent abstraction to an already existing infrastructure such as Microsoft Outlook or other desktop–server applications. The calendar agent can also represent a room and thus permit reuse of the same principles of participation and event scheduling.

Classifier

ITtalks uses a classifier, or recommender, agent that is invoked when a user enters a new talk. On the basis of the talk's abstract, the classifier returns ACM CCS numbers along with a rank, in descending order. Using a local table of classification numbers and names, ITtalks will suggest to the user 10 possible topics.

MapQuest

The MapQuest agent is a wrapper agent that lets ITtalks use external services. It interacts directly with agents (for example, the ITtalks and user agents) and accepts requests for information such as the distance between two known locations. It then phrases an appropriate request to the MapQuest system, parses the results, and generates a response. We could also generically name this agent a distance agent and use any external service (or combination of several, as needed).

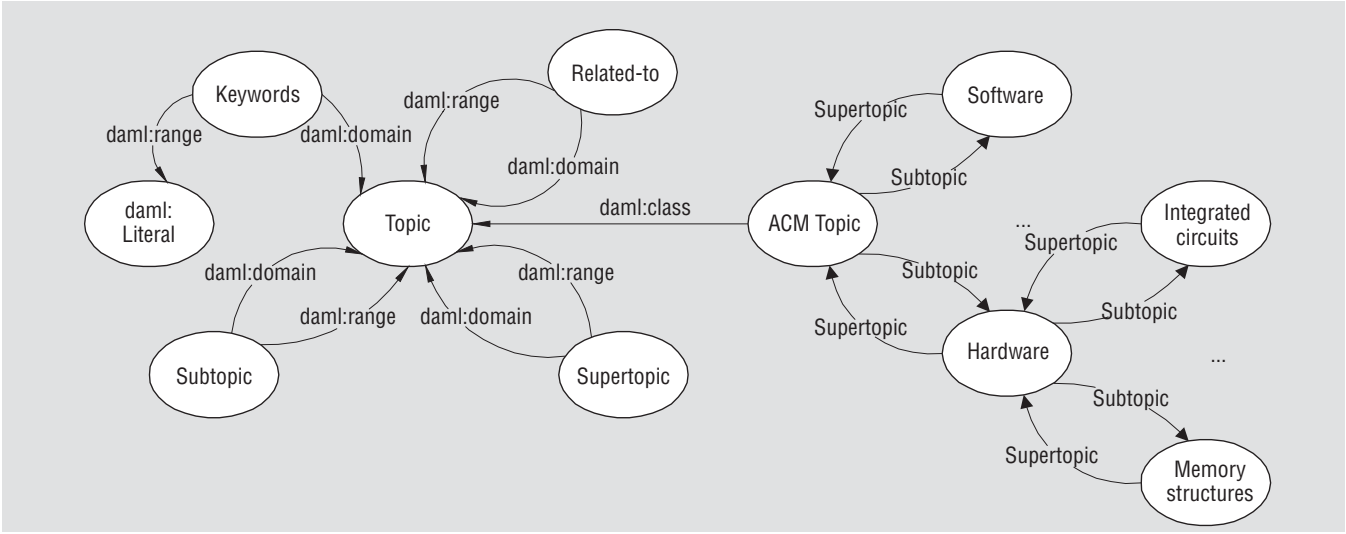

Figure 4. The ontologies that ITtalks uses are relatively simple, such as the topics ontology it uses to describe talk topics and user interests.

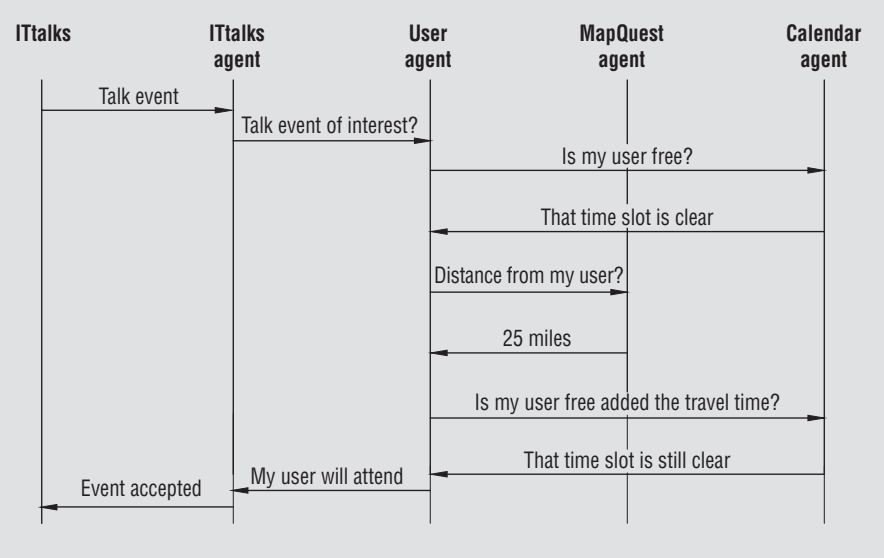

Figure 5. Agent interactions in the ITtalks agent scenario.

Scenario: Advanced features

We continue our scenario with user Jim to demonstrate more advanced interactions that ITtalks offers. Jim has registered with ITtalks and left instructions for the system to notify him when certain types of talks are scheduled.

ITtalks discovers an upcoming talk that might interest Jim. Based on his preferences, ITtalks opts to notify Jim's user agent directly via an ITtalks agent, which forwards the message using an ACL. Jim's user agent then consults Jim's calendar agent to determine his availability and the MapQuest agent for the distance from Jim's predicted location at the time of the talk (see Figure 5).

Some more sophisticated interactions, which we plan to implement, could potentially take place at this time. For example, the calendar and user agents might decide to alter Jim's schedule and contact a colleague's user agent. Or, the user agent could request more information about the speaker and event by contacting other agents or Web sites, such as a CiteSeer-based agent.[7] Finally, after making a decision, the user agent could notify the ITtalks agent, indicating that Jim will or will not attend. The ITtalks agent would then make the appropriate adjustments to the ITtalks database.

Future implementations might also allow for even more complex interactions. Say that a research group employs Jim but has limited funding. The group lets only one researcher at a time attend a particular IT event. Consequently, the user agent cannot decide on Jim's participation until it interacts with other agents representing Jim's employer and colleagues. In this case, a decision and election of a group representative requires an interaction involving an agent virtual community.

The user agent could also benefit from participating in virtual communities thanks to recommendations it obtains from other user agents. One user agent might recommend an IT event given its owner's experiences from attending a past talk of the same speaker. Another user agent might decide to share comparisons of two competing times and locations for an identical IT event. Yet another user agent might simply share its owner's intension to attend a particular IT event. Thus, each virtual community member could profit from these and other recommendations and reflect these social filtering methods in its own decisions.

Finally, future implementations might factor in smart offices.[8,9] For example, the ITtalks agent could directly contact an agent representing an IT event location. This room agent could use varying service discovery techniques[10,11] to locate a projector in the room and instruct it to download a PowerPoint presentation before the speaker's arrival. Moreover, the room agent might also try to contact additional agents in the IT event's vicinity to decrease the possible noise level from other rooms and to verify that a hallway agent has requested enough refreshments during the event.

Benefits of DAML+OIL

ITtalks benefits significantly from its use of a semantic markup language. DAML+OIL specifies the ontologies that we use extensively in our system for personal profiles and as a content language for agent communication. Without DAML+OIL, specifying topic schedules, interests, and assertions would be very difficult. As an ACL, DAML+OIL offers more flexible semantics than KIF (Knowledge Interchange Format) or other content languages that only provide syntax. The greatest benefit DAML+OIL gives ITtalks is the ability to interact with any DAML+OIL-capable agent without human supervision. Consequently, all these benefits enable more efficient interaction between the system and its users—humans or software agents. (See the "Related Work" sidebar.)

Interoperability standard

Using DAML+OIL lets us easily share ITtalks content with other DAML+OIL-aware applications and agents. In the future, we could also offer a simple extension to let arbitrary agents register and interact with ITtalks for various purposes. For example, a Centaurus room manager agent[8] could watch ITtalks for events happening in a particular room to enable better scheduling. DAML+OIL also acts as an interoperability standard, by letting other sites publish announcements marked up in our ontology, making their talks available for inclusion in ITtalks.

Agent communication language

In the future Semantic Web, agents will create, access, modify, enrich, and manage DAML+OIL documents as a way to disseminate and share knowledge. Because DAML+OIL documents will be the objects of discourse, DAML+OIL and ACLs must be successfully integrated. Agents will need to exchange DAML+OIL documents and exchange *informational attitudes* about DAML+OIL documents. Using an ACL, agents can talk about DAML+OIL documents. Thus, we hope to integrate ACL work and concepts with a universe of DAML+OIL content.

Distributed trust and belief

Agents have difficulty knowing what information sources (for example, documents, Web pages, and agents) to believe or trust in an open, distributed, and dynamic world and how to integrate potentially contradictory information. We can use DAML+OIL to support *distributed trust and reputation management*,[12,13] by forming the basis of a logic for *distributed belief transfer* that will enable more sophisticated, semantically driven rule-based techniques for information integration and fusion. Distributed trust involves authentification and access control based on the conformance of a user's credentials to a predefined security policy. We are considering *distributed belief* as a way for agents to garner required information. For example, an agent would believe that the

Related Work

ITtalks must address two general problems: generating the Semantic Web markup for talk announcements and integrating and fusing information from different systems.

ITtalks uses text classification to automatically generate topic descriptors for markup, using the Carnegie Mellon University Rainbow program (www-2.cs.cmu.edu/~mccallum/bow). Kamal Nigam and his colleagues have also written a good reference for text classification.[1]

The Lockheed Martin AeroDAML project (http://ubot.lockheedmartin.com/ubot) offers a more powerful approach to automatically generating markup for natural language documents. The project tackles the markup generation problem by combining its proprietary natural language processor, AeroText, with the DAML+OIL language for knowledge representation. Consequently, software agents can use AeroDAML to automatically annotate existing Web pages to overcome the syntactic complexity and semantic ambiguity of otherwise inaccessible information.

ITtalks uses a multiagent systems approach (http://agents.umbc.edu) to integrate information from different sources. Specifically, it wraps each information source or destination with code to enable it to act as an agent. The resulting system of agents communicates with each other using the Foundation for Intelligent Physical Agents (http://fipa.org) agent communication language and associated standards.

The WebScripter (www.isi.edu/Webscripter) system, developed at the Univ. of Southern California Information Sciences Institute, offers another approach to information fusion on the Semantic Web. WebScripter collects DAML-encoded information from multiple, heterogeneous Web sources and combines them by extracting and fusing the encoded information into reports. WebScripter comprises two software components: the report-authoring environment for defining reports, and the report instantiation component that generates new reports from DAML instances and makes them available on the Web.

Reference

1. K. Nigam et al., "Text Classification from Labeled and Unlabeled Documents Using EM," *Machine Learning*, vol. 39, nos. 2–3, 2000, pp. 103–134.

http://w3c.org/xml page would always have current information about XML.

We use DAML+OIL's expressiveness by employing it to describe the security policies, credentials, and trust relationships that form the basis of trust management. Policies specified in this way contain semantic meaning, which facilitates their integration and improved conflict resolution. Also, other applications will be able to interpret the agent's credentials (such as authorization certificates) correctly, universalizing these credentials. Similarly, describing beliefs and associating levels of trust with them is straightforward, and deducing belief is uniform for different applications and services.

Data entry support

ITtalks supports intelligent form filling, making it easier for users to enter and edit information in their profiles and talk announcements. Additionally, it provides automatic form filling when an editor tries to enter information about an entity that already exists in the knowledge base.

Entering talks

To make ITtalks successful, we must simplify the entry of new talk descriptions into the system. We address this problem using three complementary approaches: an enhanced Web interface, acceptance of marked-up announcements, and automated text extraction. DAML+OIL plays a key role in the first two and is the target representation for the third.

Enhancing the Web interface

Although our implementation of automatic form filling for entities already entered in the knowledge base does not directly use DAML+OIL, it can support a more generalized version of a form-filling assistant. This depends on

- Tagging Web form widgets with DAML+OIL descriptions of what they represent
- Capturing dependencies among data items in DAML+OIL
- Compiling these dependencies into an appropriate execution form (for example, JavaScript procedures) that can drive the Web form interface

Additionally, we plan to investigate multimodal support, where users can enter new information through keyboard or vocal input. When filling in a form, the user would speak to enter data in each field. On receipt of the voice-filled form, ITtalks would try to infer the recorded sound's meaning, obtain additional information based on the knowledge and rules stored in the ITtalks system, and present the user with a completed form for verification. This enhancement would let ITtalks provide voice-entry support for devices with limited keyboard functionality, such as PDAs or cellular phones.

Text classification

For ITtalks to filter talk announcements on topic matches, we must know the appropriate topics for each talk. Initially, we required users to manually select appropriate topic categories from the ACM CCS hierarchy. However, the user faces a daunting task of navigating a hierarchy of nearly 300 topics, many with unfamiliar meanings. Trying to select their own topics to characterize their interests can be equally daunting. Ultimately, we would like to use more than one topic hierarchy to classify both talk topics and user interests (for example, ACM CCS and Open Directory nodes), which further complicates the problem.

To address this, we built an automatic text classifier to suggest appropriate terms in a hierarchy for classifying a talk based on its title and abstract. We used the classifier package from the Bag Of Words toolkit by Andrew McCallum at Carnegie Mellon University. This library provides support for a wide variety of text classification and retrieval algorithms. We used the Naive Bayes algorithm, which is widely used in the classification literature and fairly effective, and was quick to learn the 285 classes in our test collection. We plan to use the same classification agent to suggest interest terms for users based on text found by searching their Web pages.

Additionally, we are also developing a tool for mapping multiple ontologies. Such a tool will, for example, let each user select a preferred topic ontology on the fly, and the ITtalks system will immediately adapt and present the personalized filtering results. As a prototype of the mapping tool, we chose a semiautomatic approach that lets users manually select relations (*landmarks*) among specific topics across the ontologies—for example, broader, narrower, and similar. Subsequently, the tool automatically computes the remaining mappings via the user-specified relations and our automatic text classifier, using training sets of documents for the ontologies.

The automated mapper then performs two operations. First, it accepts marked-up announcements. An easy way to enter new talk announcements is to supply an already marked-up document. The ITtalks interface lets you enter a URL for a talk announcement marked up in ontologies that ITtalks understands. Currently, these only include the native ontologies we built for this application. If talk announcements were available with semantic markup using other ontologies, we might be able to provide rules and transformation that could map or partially map the information into the ITtalks ontologies. We expect to encounter such marked-up announcements as the Semantic Web develops.

In the next step, the automated mapper automatically extracts the information. We would like to process talk announcements in plain text or HTML and automatically identify and extract the key information. This would let us fill the ITtalks database with information obtained from email or Web announcements. Others have studied the problem of recognizing and extracting information from talk announcements, mostly for use in a machine learning application.[14,15]

User profiles

We use personal profiles to help ITtalks meet individual user requirements. You can easily share this profile, and the use of DAML+OIL will allow more expressive content for schedules, preferences, and interests. The notion of a personal profile and a user agent are closely linked; a user might have one or the other, or both. The profile would likely express much of the information encoded in a user agent's knowledge base. Conversely, an agent would likely be able to answer queries about information in a profile.

Future directions

Because most users don't have personal agents, we have been developing one that you can use with this system. However, we'd like ITtalks to be able to interact with external agents of any type. The agent we are developing reasons about the user's interests, schedules, and assertions and uses the MapQuest agent to determine if a user can attend a particular talk.

We are developing a framework to use DAML+OIL in distributed trust and belief. DAML+OIL expressions on a Web page that encodes an agent's statement or other speech act are signed to provide authentication and integrity. We are working on an ontology for the description of permissions, obligations, and policies in DAML+OIL, which will support their use, exchange, and delegation by agents.

To make the data entry process more efficient, we are developing a focused Web spider, which will collect talk announcements from the source and identify key information using a text extraction system. The spider will add all found and relevant information to the ITtalks knowledge base.

In the future, all services that require personal user information should access the same user profile, eliminating the need to repeatedly enter the same information for a multitude of services. We believe that the new standard for XML Signature and Encryption under development might provide a mechanism by which users could control access to different parts of their profile.

Our system demonstrates the value of markup languages to the Semantic Web through its ability to improve Web agent functionality and to represent ontologies and ACLs. Each ITtalks Web page contains the necessary information for an agent to retrieve the page's DAML+OIL-encoded description and a responsible agent's contact information to provide more effective conversation. ITtalks thus gives each agent the ability to retrieve and manipulate all ITtalks-related information through a Web site interface or a direct agent-to-agent conversation. Hence, by combining the features of existing Web applications with the DAML+OIL-based knowledge and reasoning capabilities, we believe ITtalks presents a true Semantic Web application. ❐

Acknowledgments

The Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528 as part of the DAML program (www.daml.org) partly supported this research.

References

1. J. Hendler, "Agents and the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, Mar./Apr. 2001, pp. 30–37.

2. S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, Mar./Apr. 2001, pp. 46–53.

3. R.S. Cost et al., "Jackal: A Java-Based Tool for Agent Development," *Proc. Workshop Tools for Developing Agents* (AAAI 98), AAAI Press, Menlo Park, Calif., 1998, pp. 73–82.

4. T. Finin, Y. Labrou, and J. Mayfield, "KQML as an Agent Communication," *Software Agents*, MIT Press, Cambridge, Mass., 1997, pp. 291–316.

5. *FIPA 97 Specification Part 2: Agent Communication Language*, tech. report, Foundation for Intelligent Physical Agents, 1997; www.fipa.org (curent Jan. 2002).

6. F. Bellifemine, A. Poggi, and G. Rimassa, "Developing Multi Agent Systems with a FIPA-Compliant Agent Framework," *Software: Practice and Experience*, vol. 31, no. 2, Feb. 2001, pp. 103–128.

7. K.D. Bollacker, S. Lawrence, and C.L. Giles, "Citeseer: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications," *Proc. 2nd Int'l Conf. Autonomous Agents* (Agents 98), ACM Press, New York, 1998, pp. 116–123.

8. L. Kagal et al., "A Framework for Intelligent Services in a Mobile Environment," *Proc. Int'l Workshop Smart Appliances and Wearable Computing* (IWSAWC), 2001, pp. 195–201.

9. A. Cedilnik et al., *A Secure Infrastructure for Service Discovery and Access in Pervasive Computing*, tech. report TR-CS-01-12, Computer Science and Electrical Eng. Dept., Univ. of Maryland Baltimore County, Baltimore, Md., 2001.

10. D. Chakraborty et al., "Dreggie: Semantic Service Discovery for m-Commerce Applications," *Proc. Workshop Reliable and Secure Applications in Mobile Environment*, 2001, pp. 28–31.

11. O. Ratsimor et al., "Agents2go: An Infrastructure for Location-Dependent Service Discovery in the Mobile Electronic Commerce Environment," *Proc. ACM Mobile Commerce Workshop*, ACM Press, New York, 2001.

12. L. Kagal et al., "An Infrastructure for Distributed Trust Management," *Proc. Autonomous Agents Workshop Norms and Institutions in Multiagent Systems* (Agents 01), 2001.

13. N. Li and B. Grosof, "A Practically Implementable and Tractable Delegation Logic," *IEEE Symp. Security and Privacy*, IEEE Computer Soc. Press, Los Alamitos, Calif., 2000, pp. 27–42.

14. T. Eliassi-Rad and J. Shavlik, *Instructable and Adaptive Web-Agents That Learn to Retrieve and Extract Information,* tech. report 2000-1, Machine Learning Research Group, Dept. of Computer Sciences, Univ. of Wisconsin, Madison, 2000.

15. F. Ciravegna, "Learning to Tag for Information Extraction from Text," *Proc. 14th European Conf. Artificial Intelligence Workshop on Machine Learning for Information Extraction*, 2000.

The Authors

R. Scott Cost is a research assistant professor of computer science and electrical engineering at the UMBC. His research interests include software agents and agent communication, information retrieval, electronic commerce, and the Semantic Web. He received an AB from Colgate University, an MSE from Johns Hopkins University, and a PhD from UMBC—all in computer science. He is a member of the ACM, IEEE, and AAAI. Contact him at the Laboratory for Advanced Information Technology, Univ. of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; cost@csee.umbc.edu.

Tim Finin is a professor in the Department of Computer Science and Electrical Engineering and director of the Institute for Global Electronic Commerce at UMBC. His research interests include the applications of AI to problems in information systems, intelligent interfaces, and language processing. He received an SB in electrical engineering from MIT and a PhD in computer science from the University of Illinois. He is a former AAAI councilor and currently serves as the AAAI's representative on the Computing Research Association's board of directors. Contact him at the Laboratory for Advanced Information Technology, Univ. of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; finin@csee.umbc.edu.

Anupam Joshi is an associate professor of computer science and electrical engineering at UMBC. His research interests include networked computing, intelligent systems, data mining, the Semantic Web, content-based retrieval of video data from networked repositories, and networked high-performance computing and communications. He has a BTech in electrical engineering from the Indian Institute of Technology, Delhi, and an MS and a PhD in computer science from Purdue University. He is an associate editor of *IEEE Transactions on Fuzzy Systems*. He is a member of the IEEE, IEEE Computer Society, and ACM. Contact him at the Laboratory for Advanced Information Technology, Univ. of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; joshi@csee.umbc.edu.

Yun Peng is assistant director of the graduate program for the Department of Computer Science and Electrical Engineering at UMBC. His research interests include a variety of subjects in artificial intelligence, neural networks, and artificial life. He has a BS in electrical engineering from the Harbin Engineering Institute, China, an MS in computer science from Wayne State University, and a PhD in computer science from the University of Maryland, College Park. He is an associate editor of *Pattern Recognition* and *Computers in Biology and Medicine*. He is a member of the AAAI and the international Neural Network society. Contact him at the Laboratory for Advanced Information Technology, Univ. of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; ypeng@csee.umbc.edu.

Charles Nicholas is a professor of computer science at UMBC. He is the director of the UMBC's Center for Architectures for Data-Driven Information Processing, sponsored by the US Department of Defense. His research interests include document engineering, information retrieval, and programming languages. He received a BS from the University of Michigan, Flint, and an MS and a PhD from Ohio State University—all in computer science. He is general chair of the 2002 ACM International Conference on Information and Knowledge Management (CIKM 02). He is a member of the ACM and IEEE. Contact him at the Laboratory for Advanced Information Technology, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; nicholas@csee.umbc.edu.

Ian Soboroff is a researcher with the Retrieval Group at the National Institute of Standards and Technology. His research interests include information filtering, distributed retrieval systems, and methods for scaling current information retrieval, laboratory-style evaluations to Tbyte-sized document collections and beyond. He received his BS, MS, and PhD in computer science from UMBC. He is a member of the ACM and USENIX. Contact him at NIST, 100 Bureau Dr., Stop 8940, Gaithersburg, MD, 20899-8940; ian.soboroff@nist.gov.

Harry Chen is a PhD student and a graduate research assistant in the Department of Computer Science and Electrical Engineering at UMBC. His research interests include mobile computing and artificial intelligence. He received a 2001 PhD Research Fellowship from HP Labs. He received his BS and MS in computer science from UMBC. He is a student member of the ACM, IEEE, and AAAI. Contact him at the Laboratory for Advanced Information Technology, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; hchen4@csee.umbc.edu.

Lalana Kagal is a PhD student and graduate research assistant in the Department of Computer Science and Electrical Engineering at UMBC. Her research interests include artificial intelligence, security, and mobile computing. She has a BS and an MS in computer science from Pune University, India. Contact her at the Laboratory for Advanced Information Technology, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; lkagal1@csee.umbc.edu.

Filip Perich is a PhD student and graduate research assistant in the Department of Computer Science and Electrical Engineering at UMBC. His research interests include artificial intelligence, mobile computing, and mobile data management. He has a BA in mathematics from Washington College, Maryland. He is a member of the ACM. Contact him at the Laboratory for Advanced Information Technology, Univ. of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; fperic1@csee.umbc.edu.

Youyong Zou is a PhD student at UMBC. His research interests include Semantic Web language (DAML) reasoning, using Semantic Web languages in the Foundation for Intelligent Physical Agents, and ontology mapping. He has a BS and an MS in computer science from NanKai University, China. Contact him at the Laboratory for Advanced Information Technology, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; yzou1@csee.umbc.edu.

Sovrin Tolia is a graduate student at UMBC, pursuing his MS in computer science. His research interests include mobile data management, sensor networks, agents, and the Semantic Web. He has a BS in information systems from the Birla Institute of Technology and Science, India. Contact him at the Laboratory for Advanced Information Technology, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250; stolia1@csee.umbc.edu.

The Swoogle Semantic Web Search Engine

Web search engines like Google have made people “smarter” by providing ready access to the world's knowledge whenever they need to look up a fact, learn about a topic or evaluate opinions. The W3C's Semantic Web effort aims to make such information more accessible to computer programs by encoding it on the Web in machine understandable form. The Semantic Web languages RDF and OWL are being used to encode knowledge and to build a new layer of services, tools and applications supporting “semantic interoperability” in distributed systems. As the volume of RDF encoded knowledge on the Web grows, software agents will need their own search engines to help them find the relevant and trustworthy knowledge required to carry out their tasks. This paper discusses the general issues underlying the indexing and retrieval of RDF based information and describes Swoogle, a crawler based search engine whose index currently contains information on over a million RDF documents. Swoogle also serves human knowledge engineers by helping them to find Semantic Web ontologies, terms and instance data and to understand how and by whom they are being used. We will present some statistics derived from Swoogle's databases that characterize the current state of the Semantic Web.

Search on the Semantic Web differs from conventional Web search for several reasons. First, Semantic Web content is intended to be published by machines for machines—tools, Web services, software agents, information systems, and so forth. Although Semantic Web annotations and markup can help users find human-readable documents, there will likely be an “agent layer” between human users and Semantic Web search engines.

Second, knowledge encoded in Semantic Web languages such as RDF differs from both the largely unstructured free text found on most Web pages and the highly structured information found in databases. Such semi-structured information requires using a combination of techniques for effective indexing and retrieval. RDF and the Web Ontology Language (OWL) introduce aspects beyond those used in ordinary XML, allowing users to define terms (for example, classes and properties), express relationships among them, and assert constraints and axioms that hold for well-formed data.

Third, a single Semantic Web document can be a mixture of concrete facts, class and property definitions, logical constraints, and metadata. Fully understanding the document can require substantial reasoning, so developers must face the design issue of how much reasoning search engines can do and when they should do it. This reasoning produces additional facts, constraints, and metadata that may also need to be indexed, potentially along with the supporting justifications or provenance. Conventional search engines do not try to understand document content because the task is just too difficult and requires more research on text understanding.
Finally, the graph structure formed by a collection of Semantic Web documents differs significantly from the structure that emerges from a collection of HTML documents. This difference influences both the development of effective strategies for automatically discovering Semantic Web documents and the establishment of appropriate metrics for ranking their importance.

The Swoogle semantic web search system has been running as a web based service since the fall of 2005. The first of these papers describes the operation of the current system and the second describes some of the use cases for which is was designed and the results of using Swoogle to investigate the state of the Semantic Web.

Search Engines for Semantic Web Knowledge

We include a paper by Li Ding, and Tim Finin that appeared in the Proceedings of XTech 2006: Building Web 2.0 in May 2006. The paper provides a good overview of the Swoogle semantic web search system.

Characterizing the Semantic Web on the Web

We include a paper by Li Ding, and Tim Finin that appeared in the Proceedings of the 5th International Semantic Web Conference in November 2006. This paper explores a perspective for the Semantic Web that we call the Web aspect of the Semantic Web -- its use by independent and distributed agents who publish and consume data on the World Wide Web. To better understand this central use case, we have harvested and analyzed a collection of Semantic Web documents from an estimated ten million available on the Web. Using a corpus of more than 1.7 million documents comprising over 300 million RDF triples, we describe a number of global metrics, properties and usage patterns. Most of the metrics, such as the size of Semantic Web documents and the use frequency of Semantic Web terms, were found to follow a power law distribution.

Preprint for Tim Finin and Li Ding, Search Engines for Semantic Web Knowledge, Proceedings of XTech 2006: Building Web 2.0, Amsterdam, 16-19 May 2006.

Search Engines for Semantic Web Knowledge

Tim Finin and Li Ding

University of Maryland, Baltimore County
Baltimore MD 21250 USA

finin@umbc.edu, dingli1@cs.umbc.edu

Introduction

Web search engines like Google have made people "smarter" by providing ready access to the world's knowledge whenever they need to look up a fact, learn about a topic or evaluate opinions. The W3C's Semantic Web effort aims to make such information more accessible to computer programs by encoding it on the Web in machine understandable form. The Semantic Web languages RDF [BEC04] and OWL [DEA04] are being used to encode knowledge and to build a new layer of services, tools and applications supporting "semantic interoperability" in distributed systems. As the volume of RDF encoded knowledge on the Web grows, software agents will need their own search engines to help them find the relevant and trustworthy knowledge required to carry out their tasks. This paper discusses the general issues underlying the indexing and retrieval of RDF based information and describes Swoogle, a crawler based search engine whose index currently contains information on over a million RDF documents. Swoogle also serves human knowledge engineers by helping them to find Semantic Web ontologies, terms and instance data and to understand how and by whom they are being used. We will present some statistics derived from Swoogle's databases that characterize the current state of the Semantic Web.

Search on the Semantic Web differs from conventional Web search for several reasons. First, Semantic Web content is intended to be published by machines for machines—tools, Web services, software agents, information systems, and so forth. Although Semantic Web annotations and markup can help users find human-readable documents, there will likely be an "agent layer" between human users and Semantic Web search engines.

Second, knowledge encoded in Semantic Web languages such as RDF differs from both the largely unstructured free text found on most Web pages and the highly structured information found in databases. Such semi-structured information requires using a combination of techniques for effective indexing and retrieval. RDF and the Web Ontology Language (OWL) introduce aspects beyond those used in ordinary XML, allowing users to define terms (for example, classes and properties), express relationships among them, and assert constraints and axioms that hold for well-formed data.

Third, a single Semantic Web document can be a mixture of concrete facts, class and property definitions, logical constraints, and metadata. Fully understanding the document can require substantial reasoning, so developers must face the design issue of

how much reasoning search engines can do and when they should do it. This reasoning produces additional facts, constraints, and metadata that may also need to be indexed, potentially along with the supporting justifications or provenance. Conventional search engines do not try to understand document content because the task is just too difficult and requires more research on text understanding.

Finally, the graph structure formed by a collection of Semantic Web documents differs significantly from the structure that emerges from a collection of HTML documents. This difference influences both the development of effective strategies for automatically discovering Semantic Web documents and the establishment of appropriate metrics for ranking their importance.

The Semantic Web Model

The Semantic Web is a framework that allows publishing, sharing, and reusing data and knowledge on the Web and across applications, enterprises, and community boundaries. The W3C's approach is based on the layered set of standards shown in Figure 1. The bottom two layers provide a foundation, using XML for syntax and uniform resource identifiers (URIs) for naming. The middle three layers provide a representation for concepts, properties, and individuals based on the RDF, RDF Schema (RDFS) [BRI04] and OWL. The top-most layers, still under development, extend the semantics to represent inference rules, a logic framework, proofs, and trust.

These languages and concepts can be used in contexts other than as Web documents, including storing information in databases, exchanging data in messages and even describing the contents of networking packets. For our purposes, however, we are interested in the use of RDF to encode information on Web pages, what we will call the *Semantic Web on the Web*. In this view, the Semantic Web consists of Semantic Web documents (SWDs) typically published as Web pages encoded in XML or one of several other encoding languages. Figure 2 shows a simple SWD encoded using the RDF/XML syntax and figure 3 depicts its representation as a graph.

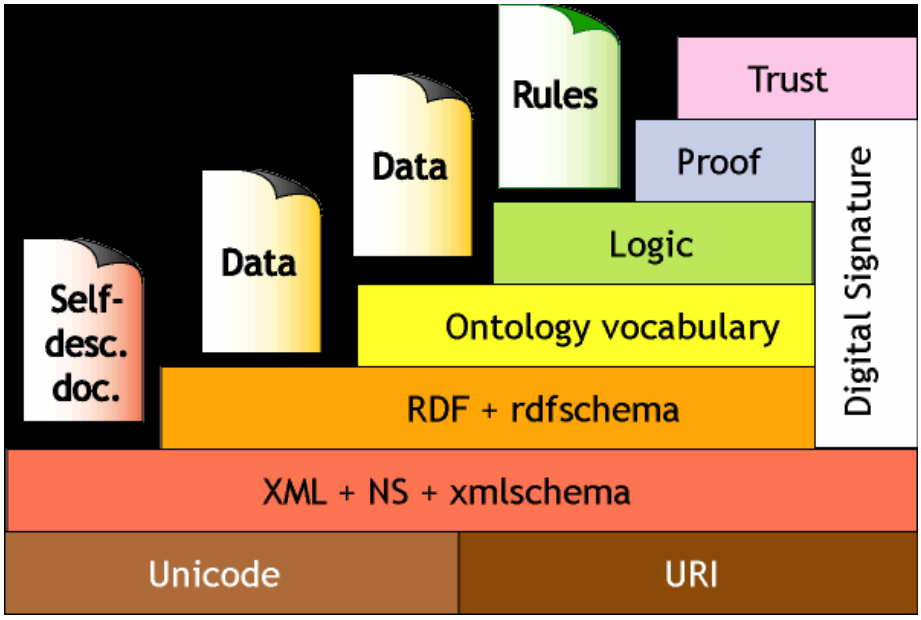

Figure 1: Tim Berners-Lee's layer cake of enabling Semantic Web standards and technologies.

```
1:<?xml version="1.0" encoding="utf-8"?>
2:<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:owl="http://www.w3.org/2002/07/owl#"
4:          xmlns:foaf="http://xmlns.com/foaf/0.1/" >
5: <foaf:Person>
6:    <foaf:name>Li Ding</foaf:name>
7:  <foaf:mbox rdf:resource="mailto:dingli1@umbc.edu"/>
8:    <owl:sameAs rdf:resource="http://www.csee.umbc.edu/~dingli1/foaf.rdf#dingli"/>
9: </foaf:Person>
10:</rdf:RDF>
```

Figure 2. *An example Semantic Web document written in RDF/XML. The SWD is available at http://ebiquity.umbc.edu/get/a/resource/134.rdf.*

Line 1 declares that this is an XML document. Lines 2-4 further define the content to be an RDF document and provide abbreviations for three common "namespaces" for RDF, OWL, and Friend of a Friend (FOAF), which defines classes and properties for describing people, their common attributes, and relations among them. The SWD's vocabulary consists of literals ('Li Ding' in line 6), URI-based resources (mailto:dingli1@umbc.edu in line 7), and anonymous resources (lines 5-9). Users assert statements using RDF triples such as the one starting at line 5, which has an anonymous resource as the subject, rdf:type as the predicate, and foaf:Person as the object. A higher level of granularity is class-instance, which RDFS's object-oriented ontology constructs offer. Lines 5-9 assert that "there is an instance of a foaf:Person having foaf:name Li Ding, foaf:mbox mailto:dingli1@umbc.edu, and this instance is owl:sameAs, identified by http://www.csee.umbc.edu/~dingli1/foaf.

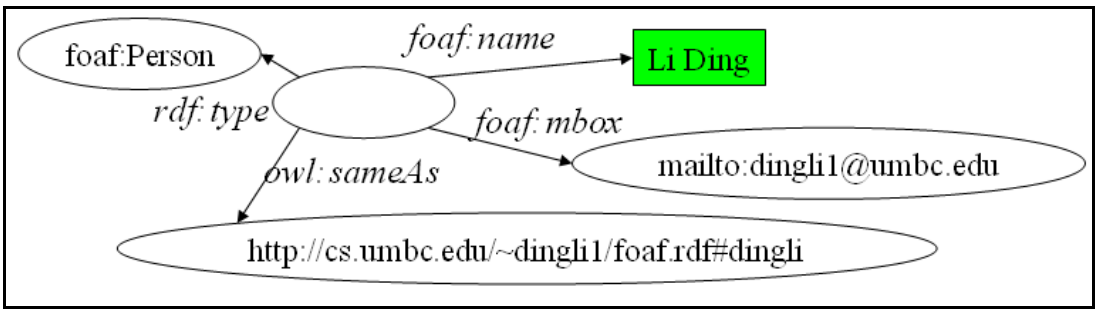

Figure 3: The RDF document in figure 2 has a simple representation as a graph.

A variation on this model is starting to become more popular – embedding Semantic Web content in HTML or XHTML pages. The microformats [KHA06] idea is being used by many as a way to include "semantic" information in HTML links. A more principled approach is being taken by a W3C working group which has developed RDF/A as a set of attributes used to embed RDF in XHTML. Currently this approach is still largely experimental. Microformats are supported by a somewhat informal standardization process and RDF/A has not yet reached the status of a recommended standard.

The Semantic Web on the Web can also be thought of as a collection of loosely federated databases that separates physical Web storage (enforced by online SWDs) from the logical representation (enforced by the RDF graph model). In this view, the Semantic Web represents a large, universal RDF graph whose parts are physically serialized by SWDs distributed across the Web. However, the formal semantics associated with Semantic Web languages support generating new facts from existing ones, while conventional databases only enumerate all facts.

Searching the Semantic Web

Search engines for both the conventional Web and the Semantic Web involve the same set of high-level tasks: discovering and harvesting documents, processing search queries from users and agents, ranking search results, caching and archiving documents, and providing human interfaces and software APIs. Figure 4 shows the high-level architecture of Swoogle. We'll discuss how we've approached each of these in turn for the Swoogle Semantic Web search engine.

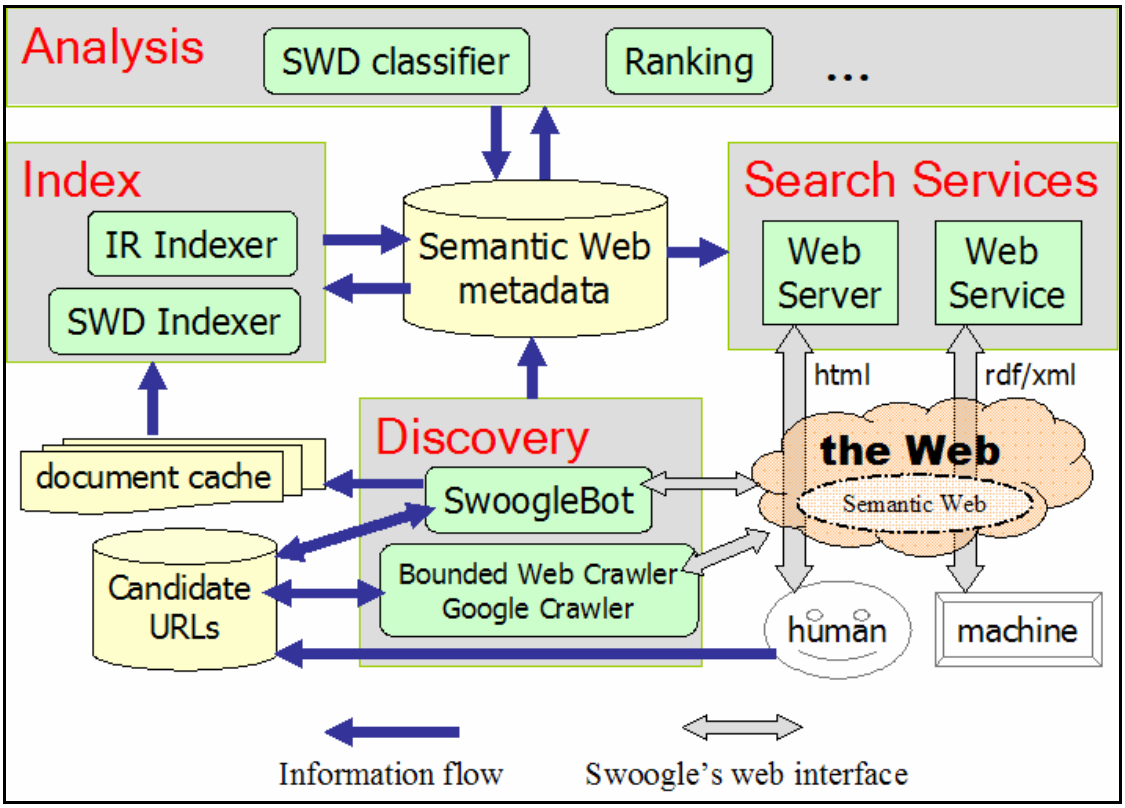

Figure 4: Swoogle's high-level architecture reveals its major components for harvesting, search, archiving and interfacing.

Harvest

Conventional search engines employ crawlers to harvest new Web documents. A typical crawler starts from a set of seed URLs, visits Web documents, and traverses the Web by following the hyperlinks found in the visited documents. The fact that the Web forms a well-connected graph and that people can manually submit new URLs make this an effective process. A Semantic Web crawler must deal with several problems. SWDs are still needles in the Web's haystack, so an exhaustive crawl of the Web is not an efficient approach. Moreover, the global SWD graph is not yet as dense and well-connected as that formed by conventional Web pages. Finally, many of the URLs found in SWDs reference conventional Web resources. Following these links can be computationally expensive, so heuristics to limit and prune candidate links are beneficial.

The URLs of SWDs be collected by manual submission or meta-search on conventional Web search engines. However, these sources usually have partial view of the Semantic Web. Conventional HTML crawling usually generates huge overhead, but it is useful in harvesting SWDs linked by certain hubs. RDF crawlers (also known as scutters) can extract links from the parsed RDF graph, but the link indicators should not be limited to rdfs:seeAlso [BID04]. Finally, using conventional Web search engines like Google to find documents with filetypes suggesting Semantic Web content has problems with both

precision and recall.

Swoogle implements a hybrid approach with several components, including a Google meta-search engine, an RDF crawler and a focused HTML crawler. Manual submission of URLs is used to bootstrap the seeds for Google and bounded HTML crawling. The two crawlers are used to automatically collect the seeding URLs of RDF crawling. The RDF crawler visits and revisits URLs to maintain an up-to-date picture of the Semantic Web, and selectively harvests new seeding URLs for itself using syntactic and semantic parsing results. The harvested SWDs are then used as training data to inductively generate new seeds for Google and HTML crawling.

Search

A search engine's core task is processing queries against the data it has indexed. While queries to Web search engines return documents, the results of a Semantic Web search query can be more or less than a document. As Figure 5 shows, a Semantic Web can aggregate data at several levels of granularity, ranging from the universal graph of all RDF data on the Web to a single RDF triple or even the constituent terms such as a URI. These levels of granularity results in the following frequently encountered search targets:

- *URIs having class/property usage by metadata.* For example, "Find classes which are immediate subclasses of foaf:Person". For Semantic Web content, these terms are analogous to words in natural language. This search helps users to generate Semantic Web data and queries.
- *URLs of SWDs by RDF graph.* For example, find documents that have a foaf:Person instance with a foaf:mbox equal to "mailto:dingli1@umbc.edu" and foaf:name equal to "li ding". This search helps users find documents on the Semantic Web that provide (partial) evidences for a given RDF graph.
- *Search for URLs of SWDs by metadata*. For example, find documents that use the OWL namespace and define properties with local names including 'before' or 'after'. This search shows users' interest in the physical storage of Semantic Web data since an SWD is the basic data transfer packet on the Web and its URL made the data addressable. This level of granularity helps improve efficiency in filtering out huge amounts of irrelevant knowledge. Some documents, such as those representing consensus ontologies, are intended for sharing and reuse. Discovering and using them is essential to achieving the goal of *semantic interoperability*.

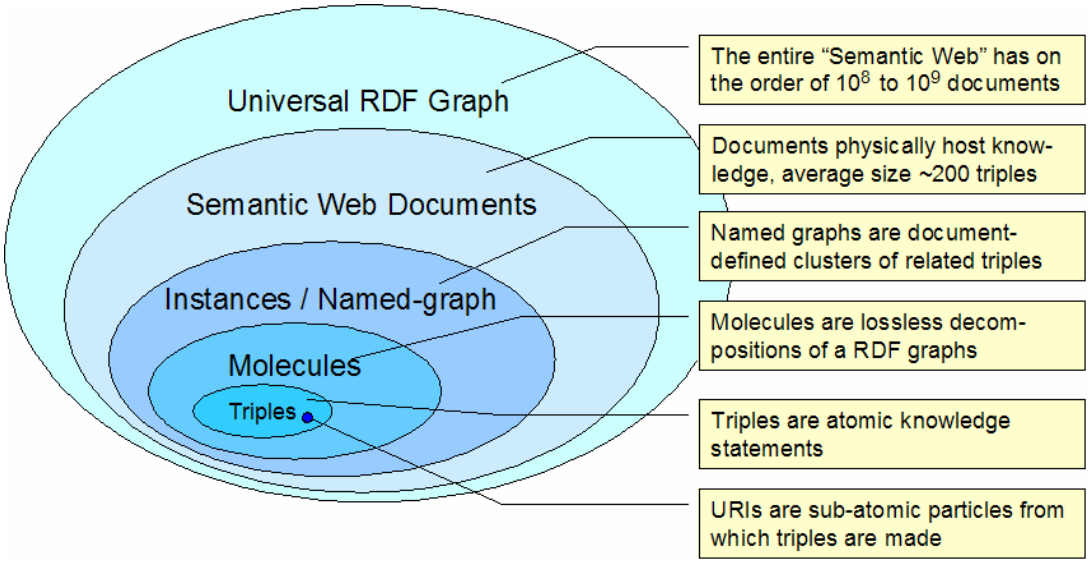

Figure 5: The granularity levels range from the universal graph comprising all RDF data on the Web to individual triples and their constituent resources and literals.

The search targets are essentially references, but the search constraints vary. The second one uses RDF graph as search constraint, and requires an RDF database storing all triples and their provenance. The first and third scenarios are similar to dictionary lookup and Web search, respectively. Using a compact metadata model can avoid the prohibitive space cost for storing all triples.

The annotation metadata of URI includes the namespace and local-name extracted from the term's URI; the literal description of the term from different SWDs. The annotation metadata of SWDs includes metadata about itself (such as document URL and last-modified time) and its content (such as terms being defined or populated and ontology documents being imported). Moreover, Swoogle maintains relational metadata that let users to combine keyword search and surfing to locate search targets.

Rank

Google was the first search engine to order its search results based in part on a Web page's "popularity" as computed from the Web's graph structure. This idea has turned out to be enormously useful in practice and is equally applicable to Semantic Web search engines. However, Google's PageRank [PAG98] algorithm, which is based on the "random surfer model", cannot be directly used in the Semantic Web for several reasons. URIs in a document are not merely hyperlinks but semantic symbols referencing classes, Semantic Web instances, ontology documents, normal Web resources, etc. Semantic Web surfing is not merely random hyperlink-based surfing but *rational surfing* that requires understanding the semantic content of documents.

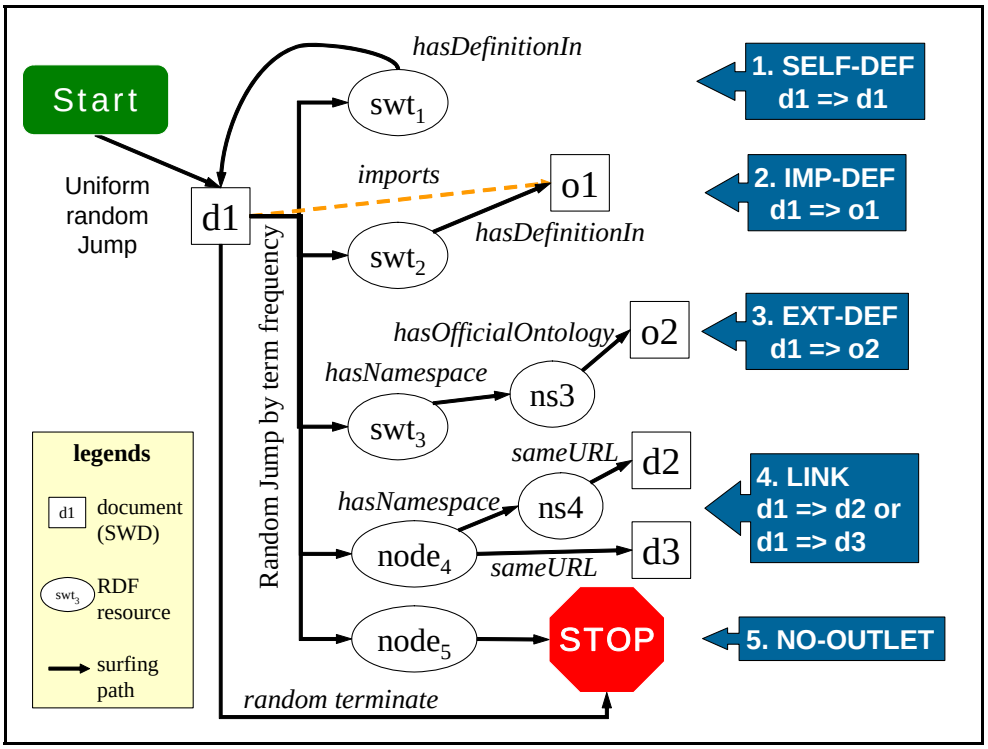

Figure 6: Swoogle's ranking algorithm for is based on a "rational surfer model" that captures how a program might access links in processing Semantic Web documents.

In order to rank the popularity of Semantic Web documents, we adopt the surfing model in which a rational surfer always recursively pursues the definition of classes and properties for complete understanding of a given RDF graph. Figure 6 illustrates the rational surfing behavior of a software agent, which unfolds as follows. The agent jumps randomly to one of the accessible SWDs with uniform probability. It either terminates surfing with constant probability or chooses one RDF node in the RDF graph of the document, and the node is chosen based on its term frequency in the N-Triples version of the document. The agent either surfs to another document or terminates surfing based on the semantics of the chosen node. Paths 1, 2 and 3 represent the agent pursuing a definition. If the node is not anonymous and is used as a class or property usage in the present document, the agent pursues its definition from the present document, the imported ontologies, or the ontology addressed by the namespace parted of the node's URI. Path 4 shows the hyper-link based surfing behavior: if the node is not anonymous and is not used as a class or property, the surfer follows the URL obtained from its URI or namespace to another Semantic Web document. Path 5 includes all cases when no further surfing path starts from the present node, e.g., the present node is literal or anonymous, or the present node's URI links to a normal Web document.

Archive

Like most search engines, Swoogle keeps a cache of the Semantic Web documents it indexes. Swoogle goes beyond this, however, in two ways. First, it also maintains a copy of each documents representation as a set of triples, a more useful form for

programs and agents. Second, and more significantly, Swoogle maintains an archive of all of the current and old versions of each Semantic Web document in its index. The resulting Semantic Web Archive can be used by researchers to study how ontologies evolve, to track the growth of documents containing RDF data or to investigate the natural life cycle of Semantic Web documents.

Applications

To explore what services a Semantic Web search engine can provide and evaluate how well Swoogle provides them, we have used Swoogle to support several applications and use cases. These projects include helping researchers find ontologies and data, semantic search over documents representing proofs, and finding and evaluating semantic associations in large graph databases.

In the NSF-supported SPIRE project [FIN04][PAR06] , a group of biologists and ecologists is exploring how to use the Semantic Web to publish, discover, and reuse models, data, and services. Researchers need to find appropriate ontologies and terms for annotating their data, and they also need resources for discovering data and services others have published.

With Swoogle's *ontology search interface*, users can search for existing ontology documents that define terms in which user-supplied keywords are the substring of their local-name. For example, to find an ontology to use for describing temporal relations, the search might use the keywords before, after and interval. Swoogle's *ontology dictionary* provides definitions of properties or classes for a given set of keywords. It can assemble and merge definitions from multiple sources, list terms sharing the same namespace or the same local-name, and list associations between classes and properties. Those associations can either be "ontological" (for example, the foaf:knows property is defined as existing between instances of foaf:person), or "empirical" (for example, applying the dc:creator property to an instance of foaf:Person). Judging the rank or popularity of terms and ontologies is also relevant. Community consensus models as reflected in ontologies tend to be ranked highly, thus searches use them more often.

Researchers are using Swoogle in conjunction with the Inference Web (IW) [PIN03] which explicitly represents proofs using the PML ontology [PIN04]. One IW component, IWSearch (http://iw4.stanford.edu/iwsearch/IWSearch/), uses Swoogle to discover newly published or updated PML documents on the Web and itself is powered by a specialized instance of Swoogle to index and search instances found in a corpus of more than 50,000 PML documents. Indexing the conclusion part of a proof NodeSet instance can lead to the discovery of additional NodeSets sharing the same conclusion as the one from the given justification tree, thus expanding the justification tree with additional proofs.

SEMDIS, an NSF project jointly conducted with researchers at the University of Georgia is also using Swoogle. This project is automating the discovery, merging, and evaluation of semantic associations in data drawn from a variety of information sources. SEMDIS augments information collected from the Semantic Web with additional data

extracted from text documents and databases [ALE06]. The result, encoded as a large RDF graph along with provenance assertions and trust information, is processed to discover and evaluate "interesting" semantic associations. SEMDIS conducts two kinds of Semantic Web searches: searching for a semantic association (i.e., a connected subgraph) in the large-scale RDF graph, and searching for additional SWDs that (partially) support a given semantic association. The first kind of search finds paths between two nodes in a graph, a common issue in RDF databases. The second is a provenance search to find a set of SWDs that (partially) imply a hypothesized semantic association. Researchers have prototyped this type of search as a RDF molecule-based approach [DIN05b].

State of the Semantic Web

How big is the Semantic Web? Is it widespread or being used by a small number of academic sites? How fast is its use growing? How many ontologies have been published and which are the most popular ones? A Semantic Web search engine like Swoogle can help answer such questions through studies on its collection of documents.

A single, metric such as the number of public RDF documents on the Web is an overly simple measure by which to chart the adoption and evolution of the Semantic Web vision. Nonetheless, it is worth computing, at least as an initial measure. For various reasons, we have prevented Swoogle from trying to find and index every published RDF document. We have, however, developed a methodology to estimate upper and lower bounds for the number of accessible Semantic Web documents using Google queries.

We estimate the lower bound using Google's filetype query feature. Since most Web documents having special filetype extensions such as "rdf" and "owl" are mainly SWDs and the keyword "rdf" is present in almost all SWDs, the Google query

> rdf filetype:rdf OR filetype:owl OR filetype:rss OR filetype:xml OR filetype:n3 OR filetype:nt

returns results that are mostly SWDs. This query, which currently returns 5.9 million results, indicates that at least this many SWDs are available on the Web and known to Google.

The upper bound is hard to estimate for several reasons. First, Google does not index all SWDs. For example, Google has indexed several hundred SWDs from the LiveJournal blogging community while LiveJournal publishes a FOAF document for each of its 10 million users. Second, our simple Google query misses some SWDs indexed by Google. Searching for "*inurl:rss –rdf -filetype:html"* finds many files that use the RSS RDF standard [RSS01]. For a rough upper bound, we use the Google query

> rdf OR inurl:foaf OR inurl:rss -filetype:html

which currently returns about 240 millions results and suggests that there are on the

order of 10^8 to 10^9 SWDs available on the Web.

Swoogle has harvested three millions candidate URLs and confirmed that 1.3 million of these are SWDs as of March 2006. Swoogle considers a document to be a SWD if it can be successfully parsed by Jena [MCB02] and produces triples. This number is less than the lower bound because Swoogle has limited access to Google's index and we have intentionally limited the number of documents collected from LiveJournal and several other sites with a large number of SWDs. We consider Swoogle's current collection, which is continually growing, to be the largest and least biased collection of Semantic Web Documents available. The following statistics are based on our analysis on these 1.3 million SWDs.

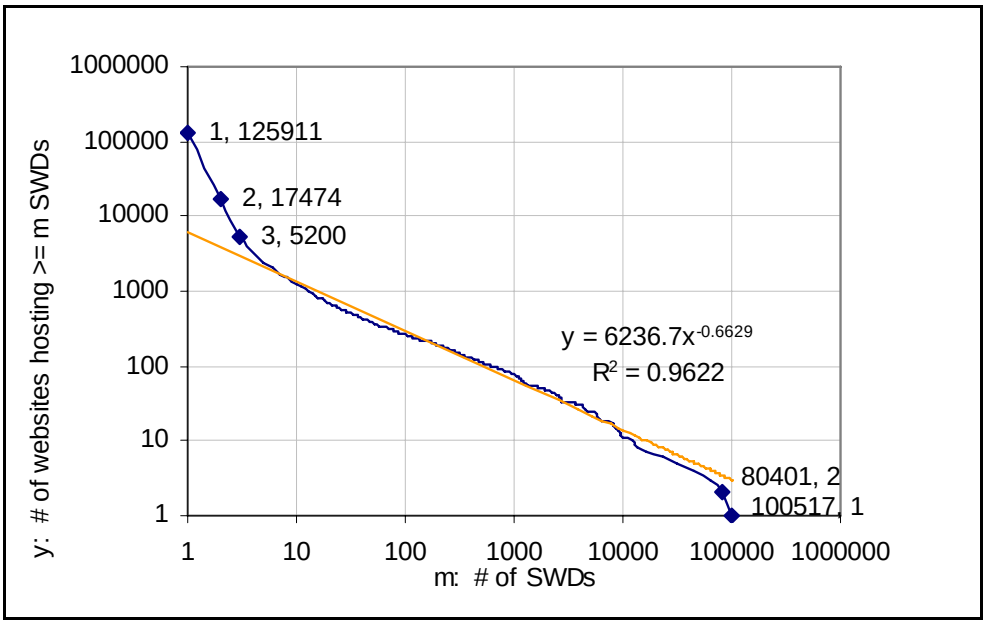

Figure 7: The cumulative distribution of the number of Semantic Web documents across Web sites follows the power law.

The 1.3 million SWDs are distributed among 125,911 websites, where a website is identified by a triple of (protocol, domain name, port number). Figure 7 shows the cumulative distribution of the number of SWDs per Web site, which generally follows power law distribution. The tail of the curve has a sharp drop when approaching 100,000 in x-axis because our crawling strategy prefers harvesting the first 10,000 SWDs from each website. The head of the curve also has a sharp drop due to virtual hosts: some content publishing websites automatically offer users a unique virtual host name under its domain.

Figure 8 shows the ten Web sites hosting the largest number of pure SWDs, i.e., RDF documents as opposed to those with embedded RDF content. The *unpinged urls* have not yet been accessed and categorized. Each of these websites is specialized in publishing one type of SWDs, such as personal profiles (e.g., FOAF documents), personal blog RSS feed documents, portable proofs (e.g., PML documents) and publication information.

Website	***# pure SWDs***	***# unpinged URLs***	***Content type***
www.livejournal.com	100,518	79,331	foaf
www.tribe.net	80,402	25,151	foaf
www.greatestjournal.com	62,453	835	foaf
onto.stanford.edu	45,278	206	pml
blog.livedoor.jp	31,741	6,733	foaf
www.ecademy.com	23,242	3,281	foaf
www.hackcraft.net	16,238	0	dc, book
www.uklug.co.uk	13,263	2	rss
users.livejournal.com	12,783	40,211	foaf
ch.kitaguni.tv	11,931	3,010	rss

Figure 8: The ten internet domains with the largest number of Semantic Web Documents.

We further count domain names, semantic web ontologies and pure semantic web documents for each top-level domain. Figure 9 shows that the .com domain has the largest contribution to Semantic Web data and Semantic Web websites while the .edu domain has the largest contribution to Semantic Web ontologies. The number of SWDs we consider to be ontologies (SWOs) is 28,564 when we filter out PML documents which contains ontological definition but are not intended to be ontologies. Swoogle considers an SWD to be an ontology if the number of its triples contributing to term definitions exceeds a threshold value.

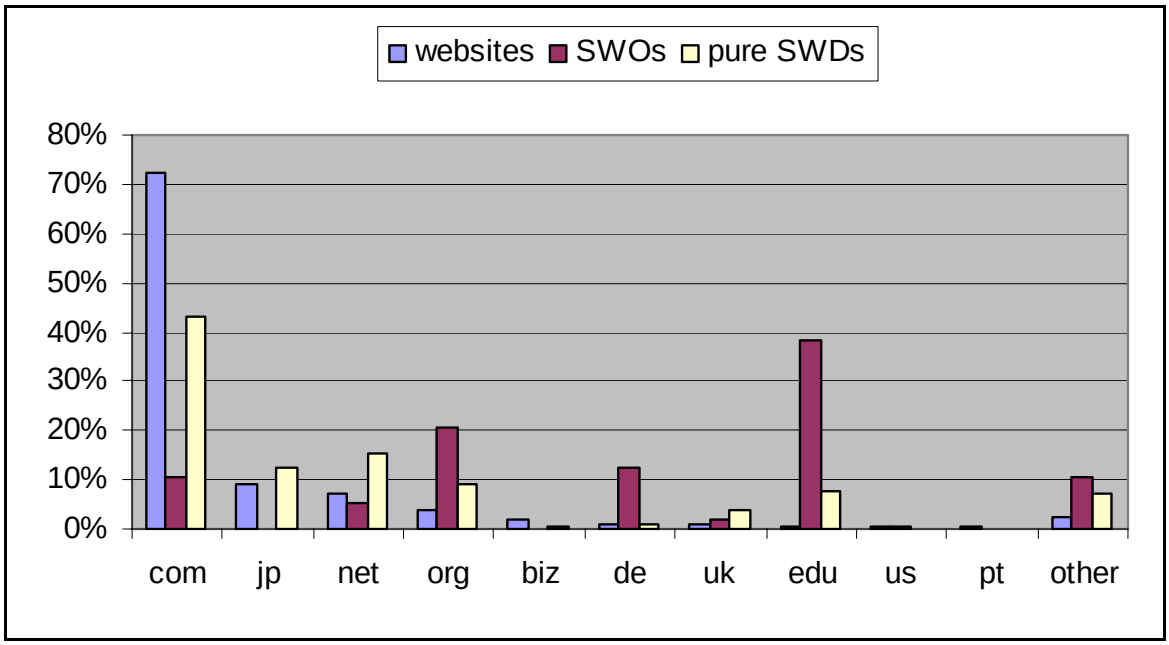

Figure 9: The distribution of Semantic Web ontologies (SWOs) and documents (SWDs) differs across the various Internet top level domains.

The size of an SWD is usually computed using the number of triples in the SWD's RDF graph. The mean size of an SWD in Swoogle's collection is 181 triples. Since many SWDs are generated by software under certain structure, some sizes are frequently observed among SWDs, for example, many PML documents have exactly 28 or 36 triples and many RSS documents have exactly 130 triples. Further investigation shows that the size of embedded SWDs are usually quite small -- 69% have exactly three triples and 96% have no more than ten triples. The size of pure SWDs varies considerably, with 60% having between five and 1000 triples.

The age of an SWD is measured by the difference between the current time and the last-modified time of the SWD. Figure 10 plots the cumulative distribution of the number of pure SWDs and SWO having been last modified before the date in X-axis. The plot excludes SWDs which does not have last-modified time specified in HTTP header and the 100K SWDs that have gone offline before March 2006.

The 'pswd' curve exhibits exponential growth; intuitively, the growth implies that either the many new PSWDs have been added to the Semantic Web and/or many old PSWDs have been updated recently. This statistics supports a promising hypothesis that the semantic web is growing rapidly. The 'swo' curve is plotted after filtered PML documents. Swoogle's data shows that the growth in the number of ontologies continues but appears to be slowing. This can be explained by an increase in the reuse of existing ontologies as the Semantic Web matures – a good sign.

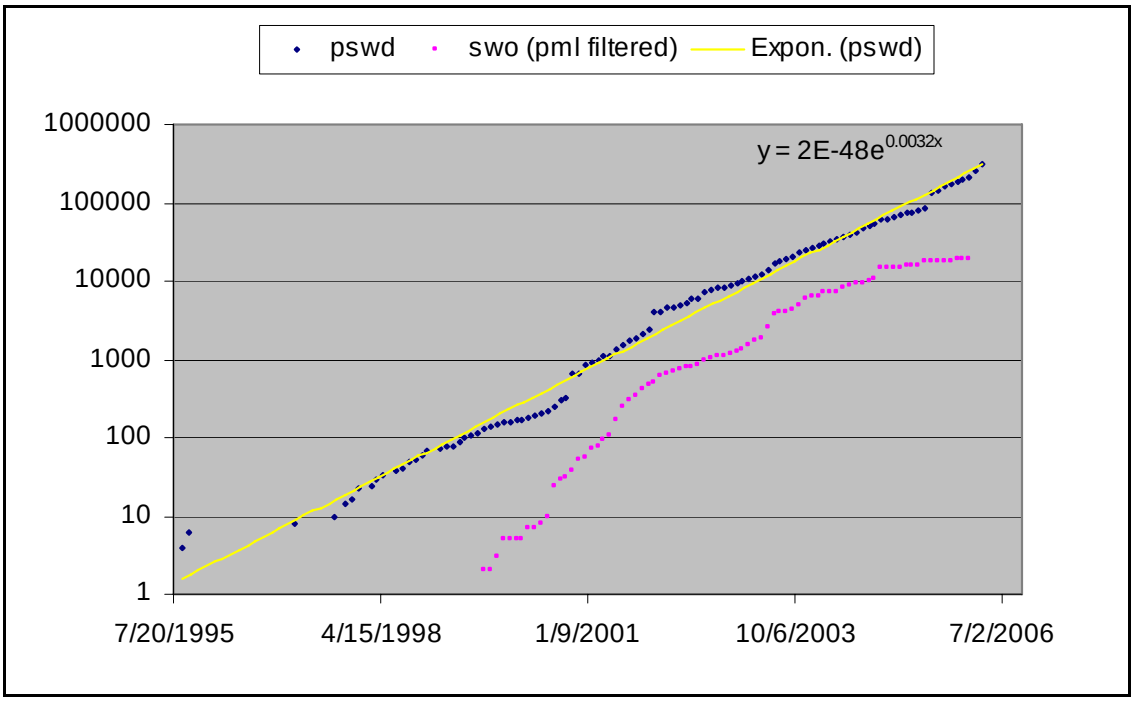

Figure 10: The distribution of the number of pure SWDs and SWOs having been last modified before the given date.

The 1.3 million SWDs contribution 237,645,189 triples (we simply sum up triples from each SWDs without merging equivalent ones), and 1,415,054 distinct Semantic Web terms (which has class/property usage in SWDs) using 11,648 namespaces.

Figure 11 shows the usage pattern of Semantic Web terms (SWTs) and introduces some interesting observations. Most SWTs (95%) are defined or referenced without being actually populated, while some SWTs (1.8%) are populated without being defined. Some SWTs (0.1%) are mistakenly used as both a class and as a property. A significant number (5.3%) are used even though they have not been defined. While some of these are undefined because they are from XMLSchma, most are due to typographic errors, misspellings, inaccessible defining documents, or other problems.
A common question posed by Semantic Web knowledge consumers is what kind of knowledge is available. One way to answer this question is by analyzing the instances found on the Semantic Web, i.e., how SWTs, classes and properties, are used to create instances and make assertions about them. We have examined the cumulative distribution of the number of SWTs associated with at least *m* instances/SWD. And found that the graphs for both classes and properties follow a power law distribution. Swoogle's collection also shows that relatively few SWTs have been used to define large amounts of data. For example, 370 classes have been used to create instances in more than 100 SWDs and 1700 classes have more than 100 instances. The same is true for properties with about 1200 properties used to assert values for instances in than 100 SWDs and about 4600 properties used in more than 100 assertions.

SWT usage pattern	***# swd***	***comments***
defined as class but not populated	1,001,571	ontology
defined as property but not populated	91,238	ontology
referenced as class only	59,289	mistakes
defined and populated as class	19,000	good practice
populated property without definition	14,266	bad practice
class defined w/o description or instances	12,929	bad practice
defined and populated as property	12,326	good practice
property defined w/o description or instances	11,291	bad practice
populated class without definition	7,761	bad practice
referenced as property only	5,672	mistakes
property defined & populated w/o description	1,940	ok practice
class defined & populated w/o description	711	ok practice
property usage w/o explicit definition	67	bad practice
class usage w/o explicit definition	449	bad practice
used/defined/referenced as class & property	1159	mistakes

Figure 11: Swoogle's collection reveals some interesting observations about how Semantic Web Terms (SWTs) are used and abused.

The ten terms most often associated with instances are shown in Figures 12 and 13, ordered by the number of documents and instances, respectively. The first number of SWDs indicates the class's popularity and the number of instance indicates the richness of instance space. Although the number of an SWT's class instances is often proportional to the number of SWDs populating the SWT, exceptions exist, for example, the WordNet Noun class has over 2 million instances but these come from just 36 large documents. Similar observations can be made about the use of properties. Swoogle's collection demonstrates that a small number of ontologies (e.g., FOAF, DC, RSS) and schema files (e.g., RDF, RDFS, OWL) dominate current Semantic Web vocabulary. Some database dumps from several authorities, such as WorldNet, NIH, CYC, IEEE, also contribute significant amount of Semantic Web instance data using giant instance document. For example, tag:govshare.info,2005:rdf/vote/option contributed 1,965,182 property instances in 14 SWDs.

Our initial studies of Swoogle's collection leads us to believe that the Semantic Web has already reached a significant size, as measured by the total number of documents and the number of sites over which they are distributed.

resource URI	*SWDs*	*instances*
http://xmlns.com/foaf/0.1/Person	467,806	11,040,981
http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq	267,603	277,608
http://purl.org/rss/1.0/channel	259,417	265,700
http://purl.org/rss/1.0/item	241,984	3,971,918
http://xmlns.com/foaf/0.1/Document	220,064	242,994
http://xmlns.com/foaf/0.1/PersonalProfileDocument	178,946	178,975
http://www.w3.org/2003/01/geo/wgs84pos#Point	85,695	107,859
http://www.w3.org/2002/07/owl#Class	62,867	1,075,220
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property	57,561	503,829
http://www.w3.org/1999/02/22-rdf-syntax-ns#List	53,726	54,491

Figure 12: The top ten SWTs ranked by the number of documents containing instances.

resource URI	**SWDs**	**instances**
http://xmlns.com/foaf/0.1/Person	467,806	11,040,981
http://purl.org/rss/1.0/item	241,984	3,971,918
http://www.cogsci.princeton.edu/~wn/schema/Noun	36	2,376,900
http://xmlns.com/wordnet/1.6/Person	2,823	1,138,374
http://xmlns.com/foaf/0.1/chatEvent	2,693	1,138,182
http://www.w3.org/2002/07/owl#Class	62,867	1,075,220
http://www.nlm.nih.gov/mesh/2004#Concept	18	734,706
http://www.daml.org/2002/02/telephone/1/areacodes-ont#Exchange	768	614,400
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property	57,561	503,829
http://www.cogsci.princeton.edu/~wn/schema/Verb	36	436,572

Figure 13: The top ten SWTs ranked by the number of instances.

Conclusions

As the Web has grown in size, search engines have become a critical component of its infrastructure, and there is an increasing need for search engines that can efficiently handle Semantic Web content. While we can't be sure what form this content will take in the future, the current standard is based on Semantic Web documents. We are continuing to use Swoogle to study the growth and characteristics of the Semantic Web and the use of RDF and OWL. We are also developing new features and capabilities and exploring how it can be used in novel applications. Many open issues remain.

One set of open problems involves scale. Techniques that work today with 5×10^6 documents may fail when the Semantic Web has 5×10^8 documents. Extending Swoogle to index and effectively query large amounts of instance data remains a challenge. Some of these problems could potentially be solved by moving away from the open source software we are using and creating custom-designed index stores and distributed systems—analogous to what Google has done for conventional Web searches. It remains to be seen, however, if that alone would suffice.

We also need to experiment with how much and where a Semantic Web search engine should reason over the contents of documents and queries. In previous work [FIN05] we experimented with expanding documents using reasoning prior to indexing. A complementary approach is to expand queries containing RDF terms [VOO94]. This is related in part to the problem of scale—the larger the collection becomes, the less efficient it is to reason over it.

Other issues involve trust and the use of local knowledge that is not part of the Semantic Web. Information encoded in RDF is now being embedded in other documents, such as PDF and XHTML documents, JPG images, and Excel spreadsheets. When techniques for such embedding become standard, we expect the growth of Semantic Web content on the Web to accelerate dramatically. This will add a new requirement for hybrid information retrieval systems that can index documents based on words as well as RDF content.

Our experience with Swoogle has given us a chance to see the growth of the Semantic Web on the Web over the past two years. The number of RDF documents has grown steadily while the number of underlying ontologies has grown more slowly, as might be expected. While the numbers are still much less than the number of conventional pages, the growth we observe makes us optimistic that the Semantic Web has a strong future.

Bibliography

[ADI06] B. Adida and M. Birbeck (eds.), RDF/A Primer 1.0 -- Embedding RDF in XHTML, W3C Working Draft, 10 March 2006. http://www.w3.org/TR/xhtml-rdfa-primer/

[ALE03] B. Aleman-Meza, C. Halaschek, I. Arpinar and A. Sheth, "Context-Aware Semantic Association Ranking," Proceedings of the First International Workshop on Semantic Web and Databases, September 2003.

[ALE06] B. Aleman-Meza, M. Nagarajan, C. Ramakrishnan, A. Sheth, B. Arpinar, L. Ding, P. Kolari, A. Joshi, and T. Finin, "Semantic Analytics on Social Networks: Experiences in Addressing the Problem of Conflict of Interest Detection", WWW 2006, Edinburgh, Scotland, May 2006.

[BEC04] D. Beckett, RDF/XML Syntax Specification (Revised), Feb. 2004; http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[BER01] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," Scientific American, May 2001, pp. 35-43.

[BID04] M. Biddulph, Crawling the Semantic Web, Proceedings of XML Europe, April 2004.

[BRI04] D. Brickley and R.V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, Feb. 2004; http://www.w3.org/TR/rdf-schema/.

[CAI04] M. Cai and M. Frank, "RDFPeers: A Scalable Distributed RDF Repository Based on a Structured Peer-to-Peer Network," Proceedings of the 13th International Conference on the World Wide Web, pp. 650-657, 2004.

[CAR04] J.J. Carroll, C. Bizer, P. Hayes and P. Stickler, Named Graphs, Provenance, and Trust, Proceedings of the 14th international conference on World Wide Web, pp 613-622, May 2005.

[DEA04] M. Dean and G. Schreiber, Web Ontology Language Reference, Feb. 2004; http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[DIN04] L. Ding et al., "Swoogle: A Search and Metadata Engine for the Semantic Web," Proc. 13th ACM Conference on Information and Knowledge Management, ACM Press, 2004.

[DIN05a] L. Ding, T. Finin, A. Joshi, Y. Peng, R. Pan and P. Reddivari, Search on the Semantic Web, IEEE Computer, volume 10, number 38, pp 62-69, October 2005.

[DIN05b] L. Ding, T. Finin, A. Joshi, Y. Peng, P. da Silva, and D. McGuinness, Tracking RDF Graph Provenance using RDF Molecules, (poster paper), Proceedings of the 4th International Semantic Web Conference, November 2005.

[DIN05c] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng and P. Kolari, Finding and Ranking Knowledge on the Semantic Web, Proceedings of the 4th International Semantic Web Conference, November 2005.

[DIN06] L. Ding, Enhancing Semantic Web Data Access, Ph.D. Dissertation, University of Maryland, Baltimore County, May 2006.

[EBE02] A. Eberhart, Survey of RDF Data on the Web, Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics, July 2002.

[FIN04] T. Finin and J. Sachs, "Will the Semantic Web Change Science?" Science Next Wave, Sept. 2004; http://nextwave.sciencemag.org.

[FIN05] T. Finin, J. Mayfield, A. Joshi, R. Cost and C. Fink, "Information Retrieval and the Semantic Web," Proceedings of the 38th International Conference on System Sciences, January 2005.

[GUO04] Y. Guo, Z. Pan, and J. Heflin, "An Evaluation of Knowledge Base Systems for Large OWL Datasets, Proceedings of the Third International Semantic Web Conference, pp. 274-288, 2004.

[KHA06] R. Kahare, Microformats: The Next (Small) Thing on the Semantic Web? IEEE Internet Computing, volume 10 issue 1, pp 68-75, January 2006.

[KLY04] G. Klyne and J.J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, Feb. 2004; http://www.w3.org/TR/rdf-concepts/.

[MCB02] Brian McBride, Jena: A Semantic Web Toolkit, IEEE Internet Computing, pp. 55-59, volume 6, issue 6, November 2002

[PAG98] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, technical report, Stanford Digital Library Technologies Project, Stanford University, 1998.

[PAR06] C. Parr, A. Parafiynyk, J. Sachs, L. Ding, S. Dornbush, T. Finin, T. Wang and A. Hollender, Integrating Ecoinformatics Resources on the Semantic Web, poster paper, Proceedings of the 15th International World Wide Web Conference, May 2006.

[PIN03] P. Pinheiro da Silva, D. McGuinness and R. McCool, "Knowledge Provenance Infrastructure," Data Eng. Bulletin, vol. 26, no. 4, 2003, pp. 26-32.

[PIN04] P. Pinheiro da Silva, D. McGuinness and R. Fikes, A Proof Markup Language for Semantic Web Services, technical report KSL04-01, Knowledge Systems Laboratory, Stanford University, 2004.

[RSS01] RDF Site Summary (RSS) 1.0, http://web.resource.org/rss/1.0/spec

[SHE05] A. Sheth et al., "Semantic Association Identification and Knowledge Discovery for National Security Applications," Journal of Database Management on Database Technology, volume 16, number 1, 2005.

[VOO94] E. Voorhees, "Query Expansion Using Lexical-Semantic Relations," Proceedings of the 17th International Conference Research and Development in Information Retrieval, 1994.

Characterizing the Semantic Web on the Web*

Li Ding[1] and Tim Finin[2]

[1] Knowledge Systems Laboratory
Stanford University, Stanford CA 94305
ding@ksl.stanford.edu

[2] Computer Science and Electrical Engineering
University of Maryland, Baltimore County, Baltimore MD 21250
finin@umbc.edu

Abstract. Semantic Web languages are being used to represent, encode and exchange *semantic* data in many contexts beyond the Web – in databases, multiagent systems, mobile computing, and ad hoc networking environments. The core paradigm, however, remains what we call the *Web aspect* of the Semantic Web – its use by independent and distributed agents who publish and consume data on the World Wide Web. To better understand this central use case, we have harvested and analyzed a collection of Semantic Web documents from an estimated ten million available on the Web. Using a corpus of more than 1.7 million documents comprising over 300 million RDF triples, we describe a number of global metrics, properties and usage patterns. Most of the metrics, such as the size of Semantic Web documents and the use frequency of Semantic Web terms, were found to follow a power law distribution.

1 Introduction

Unpacking the phrase *Semantic Web* immediately produces its two constituent concepts: it is (i) a semantic framework to represent the meaning of data that is (ii) designed for use on the Web. Most current research, both basic and applied, has focused on the first of these and largely ignored the second. An obvious lesson from the last ten years of Web-based developments is we must not underestimate the impact of the (still emerging) Web on technology and society.

Reviewing recent papers in journals and conferences one finds many on all aspects of RDF and OWL as knowledge representation languages – complexity, scalability, completeness, efficient reasoning algorithms, integration with databases, rule extensions, expressing uncertainty, human friendly encodings, etc. Developing systems and tools that use these languages for ontology engineering, visualization, manual markup, etc. is also a popular topic. Finally, application papers typically center on using RDF based representations to express the knowledge and data needed for particular problem domains, such as workflow models, action descriptions, healthcare records, policy enforcement, or user preferences. For the most part, this work touches little on issues that stem for the (initial) intended use of Semantic Web languages for publishing and using ontologies and data on the World Wide Web.

* Partial support was provided by NSF awards ITR-IIS-0326460 and ITR-IDM-0219649.

A great deal of practical work has been done, of course, on developing Web appropriate standards for the Semantic Web and harmonizing them with existing Web standards and practices. Many applications and testbeds have also focused on core Web paradigms, such as semantically enhanced Web services and policy-driven negotiation for Web resource access. Our claim is that we need more research on modeling and understanding how Semantic Web concepts and technology is and can be used on the Web. In this respect, we stand on the shoulders of those who call for *"Creating a Science of the Web"* [1].

There are also many useful and important applications of Semantic Web languages and systems that do not involve the Web. RDF and OWL are used in agent communication languages [2], instant messaging [3], and in GIS systems [4], to name just a few. We believe that the *Web aspect* of the Semantic Web remains as the common, unifying vision, one in which millions of people, agents and applications publish and consume knowledge and data using the evolving Web standards and protocols [5].

In this paper, we focus on characterizing the Semantic Web on the Web, i.e., as a collection of loosely federated knowledge bases that are semantically encoded in Semantic Web languages but are physically published and consumed on the Web by independent agents. Our work consists of three parts:

- **designing a conceptual model.** Instead of using the current model of the Semantic Web, i.e., one universal RDF graph, our new model covers both structure (RDF graphs) and provenance (Web documents and associated agents).
- **creating a global catalog.** A global catalog of online Semantic Web data has long been desired but missing; therefore, we have developed effective harvesting methods and have accumulated a significant dataset.
- **measuring data.** Using our conceptual model, we measure the collected dataset to derive interesting global statistics and implications.

Related work. While some research has tried to characterize the reach and patterns of use of the Semantic Web on the Web, they have not attempted to be systematic and have used limited datasets.

Harvesting and simple summary. A number of simple systems have been designed to find and collect RDF documents on the web, including Eberhart's RDF crawler [6], OntoKhoj [7], the DAML Crawler [8]. Several repositories for Semantic Web documents have been created and maintained using a combination of manual and automatic techniques. These include the DAML Ontology Library [9] which collected a modest number of Semantic Web documents (at most 22,000) with a limited summary of document properties such as parse error types, document size, documents per website, and namespace building usage. Additional relevant work can be found in Web characterization literature [10, 11] which studies global distributions of document properties such as the average size of web documents.

Characterizing the universal RDF graph. Gil et al. [12] analyzed the structure of a RDF graph that results from merging nearly 200 documents from the DAML Ontology Library. The dataset is too limited to be a representative of the entire Semantic Web and even the subset of ontologies on the Semantic Web.

Rating Semantic Web ontologies. Several studies have tried to measure the quality of Semantic Web ontologies, i.e. Semantic Web documents that define or contribute

to the definition of classes and properties. Most [13, 14] employ content analysis on ontologies with various foci, such as building a comprehensive evaluation framework [15], qualifying concept consistency [16, 17], quantifying the graph structure of class and property taxonomy hierarchy [18–21], and measuring the structure and the instance space of a given ontology [22]. These studies have been limited in two ways. First, they have only analyzed ontologies, which we estimate account for only about 1% of the RDF documents on the Web. Second, the empirical evaluations are based on very small datasets, typically of fewer than 30 documents.

Characterizing social networks in FOAF. One of the most successful application of RDF is the use of the FOAF ontology to encode social networks. Several studies [23–26] have analyzed large amounts of FOAF data, typically by collecting FOAF documents via specialized crawlers and then making statistical measurements on vocabulary usage and network structure. Although the evaluation datasets are large, their sources and vocabularies are limited. Most FOAF documents are obtained from a few portal websites such the *www.livejournal.com* blogging system.

Contributions. Our work is a systematic study of the semantic aspect and the web aspect of the Semantic Web. It is highlighted by contributing a new conceptual model of the Semantic Web on the Web, harvesting a significant dataset that is much larger and more diverse than other existing work, and inheriting and introducing wide spectrum of measurements for global properties on both semantic structure and knowledge provenance of the Semantic Web.

In section two of this paper we explain our conceptualization of the Semantic Web on the Web. Section three briefly illustrates our harvesting methods and evaluates the significance of harvest result. Sections four and five elaborate our metrics and findings about the global properties of the Semantic Web and section six offers some concluding remarks. In this paper, we assume, for simplicity's sake, that the following namespaces are defined: *rdf* for RDF, *rdfs* for RDF schema, *owl* for OWL, *foaf* for FOAF, *dc* for Dublin Core Element and *wn* for WordNet.

2 The Conceptual Model of the Semantic Web on the Web

The foundation for our Semantic Web characterization is the Web Of Belief Ontology which captures not only the semantic structure of RDF graph but also its provenance in terms of the Web and the agent world. This paper only covers the essential notions from the model and readers are invited to see [27] for details.

A **Semantic Web document** (SWD) is an atomic Semantic Web "data transfer packet" on the Web. It is both a Web page addressable by a URL and an RDF graph containing Semantic Web data. It can be a static or dynamic web page, for example one generated by a database query. In particular, SWDs can be divided into *pure SWDs* (PSWDs), which are completely written in Semantic Web languages, and *embedded SWDs* (ESWDs), which embed RDF graphs in their text content, e.g., HTML documents containing Creative Commons license metadata.

The *URI reference* (URIref) of an *rdfs:Resource* conveys dual semantics: (i) a unique identifier for the resource, and (ii) the Web address of the SWD defining the resource. URIrefs are widely used to merge RDF graphs distributed on the Semantic Web. A

resource's semantics depends on its usage in an RDF graph. In particular, we are interested in **Semantic Web terms** (SWTs), i.e., named resources that have **meta-usages** (being used as classes or properties) in SWDs. Six types of use are defined below and illustrated in Figure 1. For a given RDF graph, a resource X is:

- **defined as a class (DEF-C)** if there exists a triple of the form *(X, rdf:type, C)* where C is *rdfs:subClassOf rdfs:Class*. For example, *foaf:Person* is defined as a class in triple *t3*.
- **defined as a property (DEF-P)** if there exists a triple *(X, rdf:type, P)* where P is *rdfs:subClassOf rdf:Property*. For example, *foaf:mbox* is defined as a property in triple *t1*.
- **populated (or instantiated) as a class (POP-C)** if there exists a triple *(_a , rdf:type , X)* where *_a* can be any resource. For example, *rdfs:Class* has been populated as a class in triple *t3*.
- **populated (or instantiated) as a property (POP-P)** if there exists a triple *(_a , X , _b)* where *_a* and *_b* can be any resource (or literal). For example, *rdf:type* has been populated as a property in triple *t3*.
- **referenced as a class (REF-C)** if X is of type *rdfs:Class* according to the ontology constructs from Semantic Web languages except *rdf:type*. For example, *foaf:Person* is referenced as a class in triple *t2*.
- **referenced as a property (REF-P)** if X is of type *rdf:Property* according to ontology constructs from Semantic Web languages except *rdf:type*. For example, *foaf:mbox* is referenced as a property in triple *t2*.

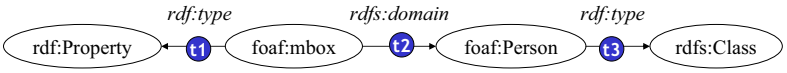

Fig. 1. This RDF graph adapted from the FOAF ontology illustrates some of the relations defined in the *Web of Belief* ontology.

Note that we may find multiple types of meta-usage of a URI in different SWDs, including some rare and undesired cases: the SWT *rdfs:subClassOf* is defined as a property by the RDFS ontology and also as a class by another SWD[3].

Two additional concepts are used studying ontologies. **Semantic Web Ontology** (SWO) is a sub-class of Semantic Web document and physically groups definitions of SWTs. An SWO is identified by containing (i) DEF-C, DEF-P, RDF-C, REF-P meta-usages or (ii) instances of *owl:Ontology*[4]. **Semantic Web Namespace** (SWN) is a sub-class of *rdfs:Resource* and logically groups SWTs and enables distributed definition (i.e., users can define the SWTs using the same SWN in different SWOs). An SWN is identified as the namespace part of an SWT.

[3] http://ilrt.org/discovery/2001/09/rdf-schema-tests/rdf-schema.rdfs

[4] The Swoogle system has experimented with different heuristics for identifying a SWD as an SWO and is currently using this very liberal one.

3 Creating a Global Catalog

In order to build a global catalog of the Semantic Web on the Web, we need to harvest publicly accessible SWDs. There are two primary difficulties: (i) SWDs are sparsely distributed on the Web and found on sites in varying density, e.g. *www.cnn.com* hosts no SWDs but *www.liverjournal.com* has millions; and (ii) Confirming that a document contains RDF content requires RDF parsing which entails high cost when done for millions of documents.

3.1 Estimating the number of online SWDs

The scale and complexity of harvesting task is dominated by the number of online SWDs, which we have estimated using the Google search engine [5] Since Google does not index all SWDs and its estimated total result is coarse, we use it to derived an *order of magnitude* estimate of the total number of online SWDs.

In theory, the search query *"rdf"* would suffice because the RDF namespace is declared by virtually all SWDs. In pracice, however, this simple Google query has two problems. First, it does not cover all indexed SWDs. For example, many RSS 1.0 files, which are RDF documents, are not matched by it. Second, it matches many documents that are not SWDs. For example the query "rdf filetype:html" identifies more than 38 million HTML documents. Based on queries run on 12 May 2006, we estimate that there are between 10^7 and 10^9 Semantic Web documents online.

- For a conservative estimate we emphasize precision and use a query where most results will be SWDs. The query "*rdf filetype:rdf*" produced 4.91M estimated matches. The constraint "filetype:rdf" was chosen because it is the most common file extension used among SWDs, and more than 75% web documents using it are SWDs [6]. This yields a conservative estimate of 10^7 SWDs.
- For an optimistic estimate we emphasize recall using a query whose results will include most online SWDs. The query "*rdf OR inurl:rss OR inurl:foaf -filetype:html*" produces about 205M results. This derives an optimistic estimate of 10^9 SWDs.

3.2 A Hybrid Semantic Web Harvesting Framework

Most existing harvesting methods are limited in significance or diversity. Conventional Web crawling approaches [6, 7] are inefficient because most hyperlinks in Web documents (including SWDs) point to conventional Web documents. Similarly, brute-force sampling, i.e., testing port 80 of reachable IP addresses [11], introduces prohibitive cost in validating millions of web documents. Meta-search based approaches [28] are limited by the inability to filter out conventional web documents from search engine results and the fact that some search engines intentionally ignore SWDs. Manual submission based approaches, such as that used for the DAML ontology library [9] and SchemaWeb [29]

[5] We have found the Google and Yahoo search engines to have the most RDF documents indexed, with Google having more than twice as many as Yahoo.

[6] Other constraints usually returns fewer results, e.g. "owl filetype:owl" returns 55K results.

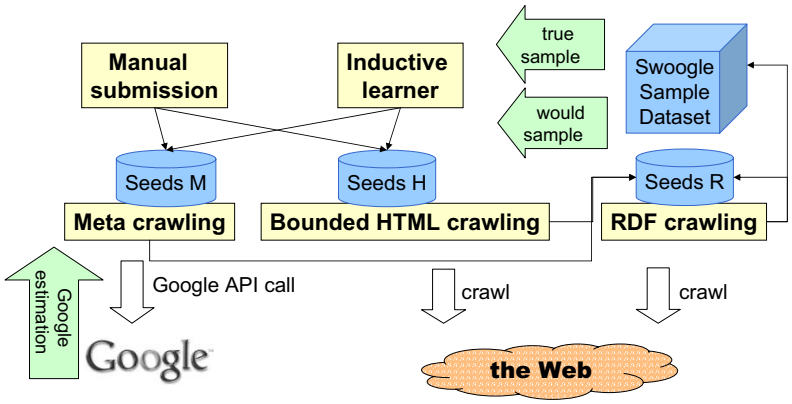

Fig. 2. The Swoogle system uses an adaptive Semantic Web harvesting framework with three different kinds of crawlers.

scale poorly and are difficult to maintain. RDF crawlers (also known as scutters[7] or Semantic Web crawlers) [30, 31] are limited because the seeding URLs (i.e., the starting points of crawling) are hard to obtain and surfing heuristics (i.e., patterns for selecting hyperlinks to SWDs) are often biased.

In order to effectively harvest as many as possible SWDs on the Web with minimum cost, we developed a automatic, hybrid Semantic Web harvesting framework [27] that integrates several harvesting methods. Figure 2 illustrates its work-flow, which has the following major steps.

1. **Bootstrapping**. Manual submissions are used to bootstrap the harvesting, providing seeds for Google-based meta-crawling and bounded HTML crawling.
2. **Google-based Meta-crawling.** *Meta crawling* [32] involves directly harvesting URLs from search engines without crawling the Web. Google is chosen because it indexes the largest number of Web documents and offers richer query constraints than others. We collect seeds from manual bootstrapping input and the *inductive learner* that selects "good" seeds from the harvested *Swoogle sample dataset*. A "good" seed is a Google query whose results contain high percentage of SWDs, e.g., most URLs returned by the query *rdf filetype:rdf* are indeed SWDs.
3. **Bounded HTML crawling.** *HTML crawling* (i.e., conventional Web crawling) harvests web documents by extracting and following hyperlinks, and is useful in harvesting clusters of SWDs on the Web. Our *bounded HTML crawling* imposes some thresholds (e.g., search depth, maximum number of URLs, and minimum percentage of SWD) to limit search space and ensure efficiency. For example, we have harvested many PML documents[8] by a bounded HTML crawl starting at http://iw.standford.edu/proofs. Again, manual submission and automated inductive learner are involved in collecting seeding URLs.

[7] See the Scutter specification at http://rdfweb.org/topic/ScutterSpec.

[8] SWDs that populate instances of the Proof Markup Language(PML) ontology (http://inferenceweb.stanford.edu/2004/07/iw.owl).

4. **RDF crawling.** The *RDF crawler* enhances conventional HTML crawling by adding RDF validation and hyperlink extraction components. It visits newly discovered URLs and periodically revisits pages to keep metadata current. For each URL, it tries to parse an RDF graph from the document using RDF parsers (e.g. Jena). If successful, it generates document level metadata and also enqueues the new discovered URLS that may link to SWDs.
5. **Inductive learner and Swoogle Sample dataset.** The sample dataset covers the metadata of the SWDs confirmed by RDF crawling. Based on the features (e.g. URL, term frequency, the source website) of harvested documents and their labels (e.g. whether they are SWD, embedded SWD or non-SWD), an automated inductive learner is used to generate new seeds for Google-based Meta-crawling and Bounded HTML crawling.

The crawler schedules its methods using the following harvesting strategies: (i) SWO harvesting has the highest priority since they are critical for users to encode and understand Semantic Web data; (ii) PSWDs are harvested with higher priorities than ESWDs because the former usually contain more Semantic Web data than the latter; and (iii) we delay harvesting URLs from websites where more than 10,000 SWDs have already been found (e.g., liveJournal) to avoid having the catalog dominated by SWDs from a few websites.

3.3 Harvesting Result and Performance

The dataset **SW06MAY** resulted from harvesting data between January 2005 and May 2006. It has 3,675,153 URLs, including 1,448,504 (40%) confirmed as SWDs, 13% confirmed as non-SWDs, 9% unreachable URLs, and 38% unpinged (not yet visited) URLs. The confirmed SWDs are from 162,245 websites[9] and contribute 279,461,895 triples. Although *SW06MAY* is much smaller than the Web with its 11.5 billion documents [33], it is much larger than any existing datasets, including:

- (2002) Eberhard [6] reported 1,479 valid SWDs out of nearly 3,000,000 URLs.
- (2003) OntoKhoj [7] reported 418 ontologies out of 2,018,412 URLs after 48-hour crawling.
- (2004) DAML Crawler reported 21,021 DAML files out of 743,017 URLs.

Significance of ontology discovery. SW06MAY contributes 83,007 SWOs including many unintended ones, such as (i) instance data with unnecessary class or property definitions or references, e.g., 55,565 (66.9%) *PML documents* from *onto.stanford.edu*, and 882 (1.1%) *semantic blog documents* from *lojjic.net*, and (ii) instance data that has unnecessary instances of *owl:Ontology*, e.g., 4,437 (5.3%) *publication metadata pages* from *www.aifb.uni-karlsruhe.de* and more *web portal metadata pages* from *ontoware.org*. Therefore, the "true" number of SWOs in SW06MAY is just 22,123 (26.7%) SWOs after removing the "unintended" ones. Moreover, this number can further reduced to 13,012 (15.7%) since there are many duplications [10].

[9] A website is uniquely identified by its domain name (host name part of a URL) but not it's IP address. Virtual hosting can result in one IP address hosting many web domains.

[10] We are currently detecting duplicate SWDs by simply comparing the md5sum of two target documents. While crude, the method is efficient and useful. For example, we have found

Significance of dataset growth. The significance of *SW06MAY* can be verified by its fast growth trend. Figure 3a shows the numbers of total URLs (*url), pinged URLs (*ping), confirmed SWDs (*swd) and confirmed pure SWDs (*pswd) discovered before the date on x-axis, and it exhibits a steady growing trend. The "ping" curve touches the "url" curve because our harvesting strategy delays harvesting URLs from websites hosting more than $10,000$ URLs until all other URLs have been visited. The increasing gap between "ping" curve and "swd" curve indicates that harvesting recall increases at the expense of the decrease of precision.

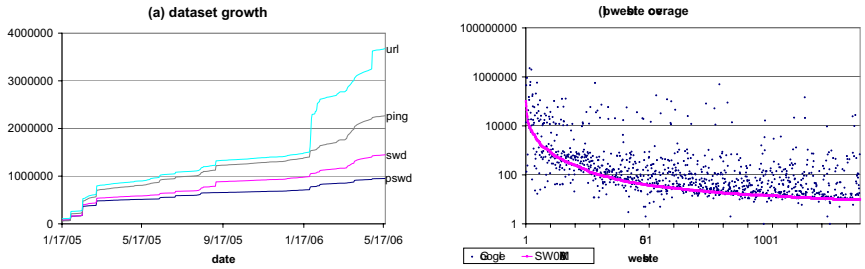

Fig. 3. The SW06MAY dataset has nearly 4M URLs collected from more than 160K sites. An analysis of the dataset demonstrates the growth in Semantic Web documents (left) and also provides evidence that our hybrid harvesting framework is sound (right).

Significance analysis on website coverage. We further evaluate the significance of *SW06MAY* by comparing its *website coverage* (i.e., the number of pure SWDs per website) with Google's estimation. In Figure 3b, each dot on the curve denotes the website coverage of one website that hosts at least ten pure SWDs. For each of the 1,355 websites in the graph, we use "Google" dots to show the optimistic Google estimation of website coverage with an additional "site" constraint, e.g., "(rdf OR inurl:foaf OR inurl:rss) -filetype:html site:www.cs.umbc.edu". The figure shows that Google's estimate, even with high variance, exhibits a trend similar to *SW06MAY*'s estimate. We conclude that the *SW06MAY* provides evidence in the basic soundness of our harvesting approach. Moreover, we suggest three causes of the variance: (i) Google's estimation may be too high since it is optimistic; (ii) The Google query site constraint searches all sub-domains of the site (e.g., site:w3.org also returns results from www4.w3.org), but *SW06MAY*'s results only return results from the specified site; and (iii) our harvesting framework may index fewer SWDs (see Google dots above the curve) because it uses far less harvesting seeds than Google and keeps a long "unpinged" list, or index more SWDs (see Google dots below the curve) because it complements Google's crawling limitation.

166 different SWDs having the same md5sum as the SWO *http://purl.org/dc/terms*. Trying to proving semantic equivalence is in general, not an option.

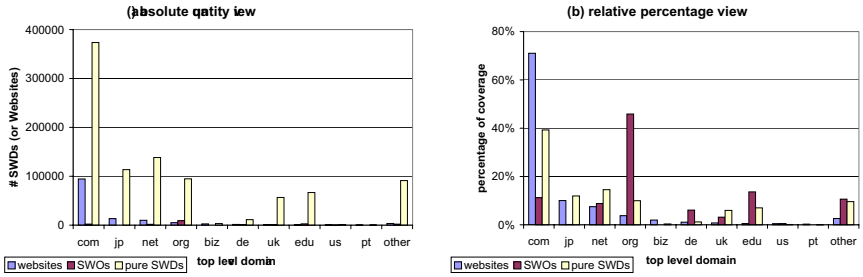

Fig. 4. An analysis of the SW06MAY dataset shows the distribution of SWDs and SWOs (after removing unintended ones) over selected top-level domains. Codes used are jp:Japan, de:Germany, uk:United Kingdom, us:United States, pt:Portugal, and other:remaining TLDs.

4 Measuring Semantic Web documents

SWD Top-level Domains. Analyzing the top-level domains (TLDs) of SWDs suggests the degree to which Semantic Web data is published by region and type of organization. Using SW06MAY we calculated the number of websites, SWDs and pure SWDs for the top ten TLDs as shown in Figure 4. The TLDs are ordered by the number of websites. Figure 4a shows that pure SWDs dominate the Semantic Web while SWOs are few in number. Figure 4b reveals several points. First, the ".com" domains have contributed the largest portion of hosts (71%) and pure SWDs (39%). Examining the data indicated two reasons: ".com" sites make heavier use of virtual hosting technology and publish many RSS and FOAF documents. Second, most SWOs are from ".org" domains (46%) and "edu" (14%). This is likely due of the deep interests in developing ontologies from academic and non-profit organizations.

SWD Source Websites. Figure 5 depicts the cumulative distribution of the number of PSWDs per website. The curves do contain skewed parts: (i) the sharp drop at the tail of curve (near 100,000 on x-axis) is caused by our harvesting strategy that delays harvesting websites after finding more than 10K SWDs; and (ii) the drop at the head of curve is due to virtual hosting technology[11]. Interestingly, *livejournal.com* is involved in both. Both curves in Figure 5 show power law distribution and the similar parameters of the two regressed equations support the conclusion that the distribution is invariant.

Table 1 lists the ten domains hosting the largest number of pure SWDs. The "content" column shows the topic of website, and the "unpinged" column indicates that we intentionally delay crawling some giant websites. SWDs from these websites are automatically generated and well inter-linked. The 6th and 9th websites are recently promoted to this list.

SWD Age. We measure an SWD's age by its last-modified time extracted from the HTTP response header. Figure 6a shows cumulative distribution of last-modified time, i.e., the number of PSWDs and SWOs with a last-modified before the date on X-axis. SWD's with no reported last-modified time are excluded. Note that the "pswd" curve exhibits an exponential distribution, indicating that many new PSWDs have been added

[11] Many social networking sites offer each user a unique virtual host name.

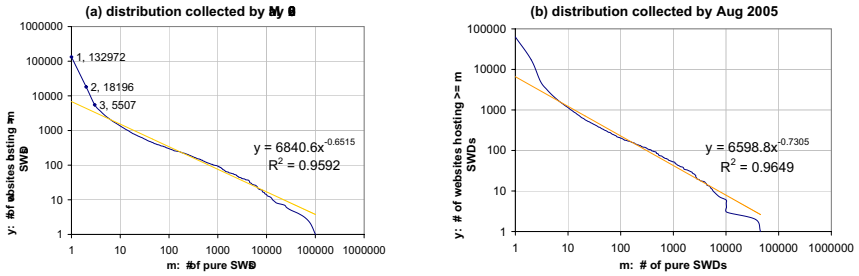

Fig. 5. Data from SW06MAY shows that the distribution of the number of websites hosting more than m pure SWDs follows a power law. The straight lines correspond to regression function with the given equations. The R^2 values close to one indicate good regressions.

rank	website	# PSWDs	# unpinged	content
1	www.livejournal.com	100,518	88,962	foaf, personal profile
2	www.tribe.net	80,402	25,234	foaf
3	www.greatestjournal.com	62,453	849	foaf
4	onto.stanford.edu	45,278	403	pml, portal proof
5	blog.livedoor.jp	31,741	12,776	foaf
6	r622-1.mpiwg-berlin.mpg.de	25,733	136	vml annotation
7	www.ecademy.com	23,242	3,308	foaf
8	www.hackcraft.net	16,238	0	dc, book annotation
9	open.bbc.co.uk	14,544	350,473	dc, BBC program annotation
10	www.uklug.co.uk	13,263	2	rss

Table 1. This table lists the ten largest source websites of pure Semantic Web documents (PSWDs) from May 2006. The *unpinged* column gives the number of URLs discovered on the site that are suspected of also being Semantic Web documents but have not yet been processed.

to the Semantic Web or that many old ones are being actively modified. The "swo" curve additionally excludes PML documents and exhibits exponential distribution with a flat tail, which we interpret as indicating a more active ontology development earlier in the time period transitioning to more reuse later.

Figure 6b shows two distributions of last-modified time collected in Aug 2005 and May 2006 respectively. The difference before August 2005 represents a loss of 155,709 PSWDs and is due to documents going offline (25%) and being updated (75%). The difference after that is caused by updated documents and newly discovered PSWDs. The non-trivial at which PSWDs go offline significantly affects the growth of Semantic Web data.

SWD Size. We measure an SWD's size as the number of triples in the SWD's RDF graph. Figure 7a shows the distribution of SWD's size, i.e., the number of SWDs having exactly m triples, and Figure 7b the corresponding cumulative distribution. Figure 7c depicts the distribution of ESWD's size. Most ESWDs are very small with 62% having exactly three triples and 97% having ten or fewer triples. These contribute significantly to the big peak in Figure 7a. Figure 7d shows the distribution of the size of PSWDs, with most (60%) having five to 1000 triples. The peaks in the curve are

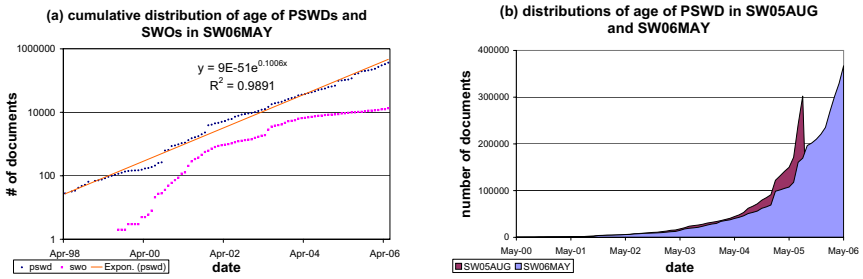

Fig. 6. Distributions of the last-modified time of PSWDs and SWOs.

caused by automatically generated SWDs which publish Semantic Web data in fixed patterns. For example, many PML documents have exactly 28 or 36 triples, and many RSS documents have exactly 130 triples[12]. The large number of SWOs with fewer than four triples are mainly RDF and OWL test documents. SW06MAY's largest SWO[13] has 1,013,493 triples and defines 337,831 classes and properties.

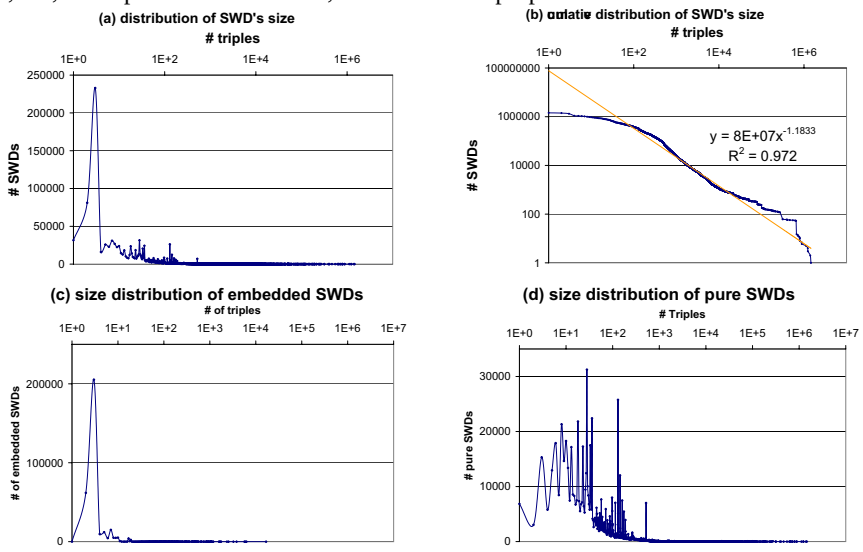

Fig. 7. The distributions of the number of triples per SWD

SWD Size Change. Updating a SWD usually result in in a change in its size. We have investigated this by tracking the size changes for different versions of an SWD. The *SW06MAY* dataset has 183,464 PSWDs that are alive (sill online) and for which we

[12] A typical RSS file has one *rss:channel* with eight triples, fifteen *rss:item* instances each with seven triples, and one *rdf:Seq* with seventeen triples connecting the *rss:channel* to the item instances.

[13] http://www.fruitfly.org/~cjm/obo-download/obo-all/ncbi_taxonomy/ncbi_taxonomy.owl

have at least three versions. For these, 37,012 (20%) lost a total of 1,774,161 triples; 73,964 (40%) gained a combination of 6,064,218 triples, and the rest 72,488 (40%) maintained their original size[14]. The statistics also show that the total number of triples keeps increasing; therefore, we hypothesize the volume of Semantic Web data is increasing.

5 Measuring Semantic Web Terms

Semantic Web Terms (SWTs) are classes and properties that are named by non-anonymous URIrefs. The *SW06MAR* dataset has 1,576,927 distinct Semantic Web terms defined with respect to 14,488 Semantic Web namespaces. We derive four SWT-usage patterns by analyzing the combination of six basic types of meta-usages.

- Only a few classes (1.5%) and properties (1.0%) have both explicit definitions and instances.
- Most SWTs (95.1%) have no instances, and some SWTs (2.2%) have no definitions.
- Some SWTs (0.08%) mistakenly have both class and property meta-usage.
- Some SWTs (0.08%) only have REF-C or REF-P meta-usages. While some are *XMLSchma* terms and not RDF, others appear to be due to errors or misuse.

SWT Definition Complexity. A simple way to measure the complexity of a SWT is to count the number of triples used to define it. Figure 8a shows the cumulative distribution of the size of SWT definitions in the curve labeled "all". This follows a power law distribution with the deviations at the head and tail reflecting a preference for defining SWTs using a manageable number of triples, two to ten triples in most cases. Terms that can be defined in just a few triples are not very useful, and the definitional size of complex terms can be reduced by defining and using auxiliary definitions. One observed definition has nearly 1000 triples[15]. We've divided definitional triples into two classes: annotation and relation triples, whose *rdf:object*s are *rdfs:Literal*s and *rdf:object*s, respectively. Note that relation triples are more common. We also noticed that 104,152 SWTs have been defined in more than one SWOs.

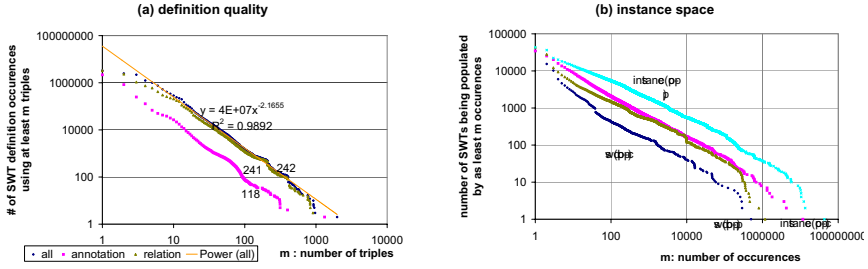

Fig. 8. The cumulative distribution of meta-usages of SWT

[14] Most of the PSWDs maintaining their size are RSS documents.

[15] The SWD http://elikonas.ced.tuc.gr/ontologies/DomainOntologies/middle_ontology defines the *MOSemanticRelationType* class using 973 triples.

SWT Instance Space. Since Semantic Web data include both definitions and instance data, we measure the instance space of the Semantic Web by counting POP-C and POP-P meta-usages of SWTs[16]. Figure 8b shows the cumulative distribution of the number of SWTs populated as a class (or property) by at least m instances (or SWDs). All four curves follow a power law distribution. For both classes and properties, most are defined but never directly used. Only 423 classes have been instantiated by more than 100 SWDs and just 2,115 have more than 100 instances. The number of properties used is somewhat higher, with 1,489 SWTs used to define data in more than 100 SWDs and 5,404 properties used in more than 100 assertions.

Table 2 lists popular classes and properties. The number of an SWT's class-instances is usually proportional to the number of SWDs populating the SWT; however, exceptions exist. For example, while the *wn:Noun* class has significant number of instances, they are mostly in a few huge SWDs. In general, the Semantic Web's instance space is dominated by three categories: (i) instances of meta-ontologies such as OWL, (ii) instances of a small number of very popular ontologies such as DC, FOAF, and RSS; and (iii) instances from giant data files, such as WordNet and National Library of Medicine's Medical Subject Headings (MeSH) ontology.

resource URI	**#swd**	**#instance**
Most instantiated classes ordered by #swd		
`http://xmlns.com/foaf/0.1/Person`	499,671	11,686,519
`http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq`	290,321	308,907
`http://purl.org/rss/1.0/channel`	282,677	289,160
`http://purl.org/rss/1.0/item`	259,220	4,277,868
`http://xmlns.com/foaf/0.1/Document`	223,510	247,311
Most instantiated classes ordered by #instance		
`http://xmlns.com/foaf/0.1/Person`	499,671	11,686,519
`http://purl.org/rss/1.0/item`	259,220	4,277,868
`http://www.cogsci.princeton.edu/~wn/schema/Noun`	56	3,697,400
`http://www.w3.org/2002/07/owl#Class`	68,053	1,795,941
`http://www.nlm.nih.gov/mesh/2004#Concept`	38	1,551,046
Most instantiated properties ordered by #swd		
`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`	1,170,975	43,291,848
`http://purl.org/dc/elements/1.1/title`	801,254	13,448,548
`http://xmlns.com/foaf/0.1/mbox_sha1sum`	462,198	2,633,739
`http://purl.org/dc/elements/1.1/description`	453,826	2,874,327
`http://www.w3.org/2000/01/rdf-schema#seeAlso`	432,288	12,330,223

Table 2. This table shows the most popular Semantic Web classes and properties based on the number of Semantic Web documents (SWDs) that use them and, for classes, also on the number of immediate instances.

RDFS and OWL usage. To what degree does the current Semantic Web make use of RDFS and OWL? One simple way of addressing this question is to examine the

[16] Since no RDFS or OWL inferencing is done, the statistics reflect immediate class instances.

number of SWDs that use the RDFS and OWL namespaces. The OWL namespace has been declared by 112,870 SWDs (8%) and actually used by 108,059 (7%). The RDFS namespace enjoys more use, being declared by 677,049 (47%) and used by 537,614 (37%) SWDs.

What about their terms? Not surprisingly, *owl:Class* is the most used term from the OWL namespace with 1,795,941 instantiations in 68,053 SWDs. Contrasting this with *rdfs:Class*, which has 327,485 instantiations by 8,572 SWDs, seems to suggest that OWL is being more heavily used than RDFS. However, the relationship is not so simple. When examining properties, rdf:Property has 529,052 immediate instantiations from 58,598 SWDs, considerably more than the OWL property terms owl:ObjectProperty (169,885 assertions in 8,041 SWDs) and owl:DatatypeProperty (48,386 assertions in 4,557 SWDs).

For RDFS and OWL properties, the most used properties is *rdf:type*, followed by some annotation properties such as *rdfs:seeAlso* and *rdfs:label*. Among those properties that are used as ontology constructs, *owl:sameAs* and *rdfs:subClassOf* are the most used. We also noticed significant use of two OWL equality assertions: *owl:sameAs* (279,648 assertions in 17,425 SWDs) and *owl:equivalentClass* (69,681 assertions in 4,341 SWDs). Their common use may be an indication of increased ontology alignment. We have found limited use of properties that require OWL DL or OWL FULL reasoning support. The most common one in our dataset was *owl:unionOf* which is used in only 2,527 SWDs.

Instantiation of *rdfs:domain*. Semantic Web data is published asynchronously by autonomous and distributed agents which may use, and misuse, a variety of ontologies. Given enough data, we can attempt to reverse-engineer the definitions of classes and terms introduced by ontologies. Consider instances of the *rdfs:domain* relation which associates a class with properties that describe its instances. We have observed 111,071 unique instantiations of *rdfs:domain*, and the number of instantiations that have been observed in at least m instances (or SWDs), again, follows a power law distribution.

The highly instantiated *rdfs:domain* relations are mainly from popular instance space such as FOAF and RSS documents. An interesting observation is that *rdfs:seeAlso* property has been frequently used as *instance property* of *foaf:Person*. This corresponding definition cannot be found in the RDFS or FOAF ontologies although it has been informally mentioned in FOAF specification. The popularity of instantiation is usually determined by the number of SWDs that has the instantiation; moreover, we also noticed a popular instantiation – the domain of *wn:wordFrom* is *wn:Noun* which has over 6.5 million occurrences in only 56 SWDs.

We can use data on the instantiations of *rdfs:domain* relation to derive the most used properties of a given class. For example, for immediate *foaf:Person* instances, the most common properties used are *foaf:mbox_sha1sum* (461,922 SWDs), *rdfs:seeAlso* (385,516), and *foaf:nick* (361,901). We can also find strong co-occurrence association among properties of a class. The properties *geo:lat* (85,742) and *geo:long* (85,741) are virtually always used together in modifying a class *geo:Point*. This kind of information can be used to help publishers choose a good set of properties, which may be from different ontologies, for a given class. Moreover, we can use such information in on-

tology revision, e.g., adding the missing *rdfs:domain* definition or revise incompatible definition.

6 Conclusions

The Semantic Web is not just one universal RDF graph but a federated collection documents distributed on and accessed via the World Wide Web. It must be studied from both the *Web perspective* and the *semantic perspective*. In order to characterize the Semantic Web on the Web and guide Web-scale data access, we estimated the size of the Semantic Web using Google, implemented a hybrid framework for harvesting Semantic Web data, and measured the results to answer questions on the Semantic Web's current deployment status.

The statistics where characterized by power law distributions and "complex system" behavior in many cases and, in general, support several conclusions about the emerging Semantic Web. (i) Semantic Web data is growing steadily on the Web even when many documents are only online for a short-while. (ii) The space of instances is sparsely populated since most classes ($>$97%) have no instances and the majority of properties ($>$70%) have never been used to assert data. (iii) Ontologies can be induced or amended by *reverse engineering* the instantiations of ontological definition in instance space [27].

Our work raises question about the current paradigm for ontologies and URIrefs. Is the concept of an "ontology" as a collection or container for Semantic Web terms needed or even useful? An ontology object encourages self consistency but introduces some limitations as well. Recent work on ontology partitions argues against large, monolithic ontologies in favor of having many interconnected components. We might even eliminate namespaces as boundaries. For example, the Dublin Core Element ontology has been widely used together with terms from many other semantic web ontologies. Another debatable item is the URIref. We use triples to annotate an URIref that is an identifier of a resource. Multiple RDF graphs from different documents describing the same URIref can introduce inconsistency. Integrating these definitions may encounter several questions: (i) are URIrefs good enough for grouping the triples describing it; (ii) can we ensure that all of the graphs are accessible to consumers; and (ii) should all be used or should some be rejected as untrustworthy.

References

1. Berners-Lee, T., Hall, W., Hendler, J., Shadbolt, N., Weitzner, D.J.: Creating a science of the web. Science **313** (2006) 769–771
2. Zou, Y., Finin, T., Ding, L., Chen, H., Pan, R.: Using Semantic web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment. In: Proceeding of the 5th International Conference on Electronic Commerce. (2003)
3. Franz, T., Staab, S.: Sam: Semantics aware instant messaging for the networked semantic desktop. In: Proceedings of the ISWC 2005 Workshop on The Semantic Desktop - Next Generation Information Management and Collaboration Infrastructure. (2005)
4. Visser, U., Stuckenschmidt, H., Schuster, G., Vogele, T.: Ontologies for geographic information processing. Computers and Geoscience **28** (2002) 103–117

5. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284** (2001) 35–43
6. Eberhart, A.: Survey of rdf data on the web. Technical report, International University in Germany (2002)
7. Patel, C., Supekar, K., Lee, Y., Park, E.K.: OntoKhoj: a semantic web portal for ontology searching, ranking and classification. In: WIDM'03. (2003)
8. Dean, M., Barber, K.: Daml crawler. http://www.daml.org/crawler/ (August 2006) (2002)
9. DAML: The DAML ontology library. http://www.daml.org/ontologies/ (August 2006) (2004)
10. Pitkow, J.E.: Summary of www characterizations. Computer Networks **30** (1998)
11. Lawrence, S., Giles, C.L.: Accessibility of information on the web. Nature **400** (1999)
12. Gil, R., Garca, R., Delgado, J.: Measuring the semantic web. SIGSEMIS Bulletin **1** (2004)
13. Hartmann, J., Sure, Y., Giboin, A., Maynard, D., del Carmen Surez-Figueroa, M., Cuel, R.: Methods for ontology evaluation. Technical report, University of Karlsruhe (2004)
14. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: A theoretical framework for ontology evaluation and validation. In: Proc. of the 2nd Italian Semantic Web Workshop. (2005)
15. Lozano-Tello, A., Gomez-Perez, A.: ONTOMETRIC:a method to choose the appropriate ontology. Journal of Database Management **15** (2003)
16. Welty, C.A., Guarino, N.: Supporting ontological analysis of taxonomic relationships. Data Knowledge Engineering **39** (2001)
17. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging owl ontologies. In: WWW'05. (2005)
18. Magkanaraki, A., Alexaki, S., Christophides, V., Plexousakis, D.: Benchmarking RDF schemas for the semantic web. In: ISWC'02. (2002)
19. Supekar, K., Patel, C., Lee, Y.: Characterizing quality of knowledge on semantic web. In: FLAIRS'02. (2002)
20. Alani, H., Brewster, C.: Ontology ranking based on the analysis of concept structures. In: K-CAP'05. (2005)
21. Yao, H., Orme, A.M., Etzkorn, L.: Cohesion metrics for ontology design and application. Journal of Computer Science **1** (2005)
22. Tartir, S., Arpinar, I.B., Moore, M., Sheth, A.P., Aleman-Meza, B.: Ontoqa: Metric-based ontology quality analysis. In: Proc. of Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources. (2006)
23. John C. Paolillo and Elijah Wright: The Challenges of FOAF Characterization. In: Proc. of the 1st Workshop on Friend of a Friend, Social Networking and the (Semantic) Web. (2004)
24. Grimnes, G.A., Edwards, P., Preece, A.: Learning meta-descriptions of the foaf network. In: ISWC'04. (2004)
25. Mika, P.: Social Networks and the Semantic Web: An Experiment in Online Social Network Analysis. In: Proc. of International Conference on Web Intelligence. (2004)
26. Ding, L., Zhou, L., Finin, T., Joshi, A.: How the semantic web is being used:an analysis of foaf. In: Proceedings of the 38th International Conference on System Sciences. (2005)
27. Ding, L.: Enhancing Semantic Web Data Access. PhD thesis, UMBC (2006)
28. Zhang, Y., Vasconcelos, W., Sleeman, D.: Ontosearch: An ontology search engine. In: Proc. of 24th Conf. on Innovative Techniques and Applications of Artificial Intelligence. (2004)
29. Lindesay, V.: The schemaweb repository. http://www.schemaweb.info/ (August 2006) (2005)
30. Biddulph, M.: Crawling the semantic web. In: XML Europe. (2004)
31. Apsitis, K., Staab, S., Handschuh, S., Oppermann, H.: Specification of an RDF Crawler. http://ontobroker.semanticweb.org/rdfcrawl/help/specification.html (March 2006) (2005)
32. Sherman, C.: Metacrawlers and metasearch engines. http://searchenginewatch.com/links/-article.php/2156241 (March 2006) (2004)
33. Gulli, A., Signorini, A.: The indexable web is more than 11.5 billion pages. In: WWW'05 (poster). (2005)

SweetJess and SweetRules

SweetJess (Semantic WEb Enabling Technlogies for Jess) is a system we developed for Semantic Web rules in Jess in 2001 and 2002 using DAML+OIL. SweetRules was developed in 2003-2005 as a powerful integrated set of tools for semantic web rules and ontologies, revolving around the RuleML and OWL.

SweetJess: Inferencing in Situated Courteous RuleML via Translation to and from Jess Rules

The SweetJess approach made four main contributions. First, it showed how to translate from rules in the Situated Courteous Logic Programs (SCLP) knowledge representation, syntactically encoded in RuleML, into Jess rules, and likewise to translate from a broad but restricted case of Jess rules into SCLP RuleML. SCLP is expressively powerful and features prioritized conflict handling and procedural attachments. The translation applies to a broad but restricted case in each direction, and preserves semantic equivalence – i.e., for a given rule base, the same conclusions are entailed. Second, it gave an architecture to perform (a broad case of) SCLP RuleML inferencing using the Jess rule engine. Third, rather straightforwardly, we have developed a DAML+OIL ontology for (SCLP) RuleML itself. The resulting syntax for RuleML is called "DamlRuleML"; the DAML+OIL is simply used as "syntactic sugar" for encoding of RuleML. Fourth, the translation newly enables bi-directional inter-operability, via RuleML, between Jess — a "reactive" rule system — and multiple other heterogeneous rule systems, including Prologs and relational database systems ("derivational" rule systems), for which translation to RuleML has already been shown and among which there are several existing translation tools (e.g., our SweetRules system). It thereby moves a discernible step closer to the Semantic Web's vision of wide knowledge sharing and integration among intelligent applications, e.g., where rules are already often deployed for e-business policies and workflow.

SweetRules and SweetDeal

SweetRules is a uniquely powerful integrated set of tools for semantic web rules and ontologies, revolving around the RuleML (Rule Markup/Modeling Language) emerging standard for semantic web rules, and supporting also the closely related SWRL (Semantic Web Rule Language), along with the OWL standard for semantic web ontologies, which in turn use XML and, optionally, RDF. (SWRL rules are essentially an expressive subset of RuleML rules.) SweetRules supports the powerful Situated Courteous Logic Programs extension of RuleML, including prioritized conflict handling and procedural attachments for actions and tests. SweetRules' capabilities include semantics-preserving translation and interoperability between a variety of rule and ontology languages (including XSB Prolog, Jess production rules, HP Jena-2, and IBM CommonRules), highly scaleable backward and forward inferencing, and merging of rulebases/ontologies.

Procedural attachments can even be WSDL Web Services. SweetRules' pluggability and composition capabilities enable new components to be added relatively quickly. Implemented in Java, SweetRules has a compact codebase (~20K lines of code total for several dozen tools).

Working Paper, version of May 2, 2003

SweetJess: Inferencing in Situated Courteous RuleML via Translation to and from Jess Rules

Benjamin N. Grosof[1], Mahesh D. Gandhe[2], and Timothy W. Finin[3]

[1] MIT Sloan School of Management,
50 Memorial Drive, Cambridge, MA 02142, USA
bgrosof@mit.edu
http://www.mit.edu/~bgrosof
[2] IBM, Business Integration Solutions
577, Airport Blvd., Suite 800, CA 94410 ,USA
maheshg@us.ibm.com
www.cs.umbc.edu/~mgandh1
[3] Department of Computer Science,
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore MD 21250 ,USA
finin@cs.umbc.edu
www.cs.umbc.edu/~finin

Abstract. We describe the innovative design of our prototyped SweetJess tool for RuleML inferencing. Our first contribution is to give a new, implemented translation from a broad but restricted case of SCLP RuleML into Jess rules, and an inverse translation from a broad but further restricted case of Jess rules into SCLP RuleML. SCLP stands for the Situated Courteous Logic Programs knowledge representation; this is expressively powerful and features prioritized conflict handling and procedural attachments. The translation is intended to preserve semantic equivalence – i.e., for a given rulebase, to entail the same conclusions. The translation often preserves semantic equivalence; in current work, we are developing formal guarantees for the equivalence, including necessary expressive restrictions in each direction. Our second contribution, building upon the translation, is a new, implemented architecture to perform (a broad case of) SCLP RuleML inferencing using the Jess rule engine. Our approach translates from SCLP RuleML rules into Jess rules, runs the Jess rule engine to generate conclusions (and actions), and then translates the concluded Jess facts back into SCLP RuleML. Our third new contribution is to enable bi-directional implemented inter-operability, via RuleML, between several other heterogeneous rule systems (e.g., XSB Prolog and IBM CommonRules) and Jess. For example, to our knowledge, this is the first tool to enable inter-operability between a Prolog and any "production"/"reactive" rule system descended from OPS5 heritage. The prototype implementation of SweetJess is publicly available for Web download.

1 Introduction and Overview

The overall problem we address is how to enable inter-operability between heterogeneous rule systems (including relational database systems as an important special case), and between heterogeneous intelligent applications that make use of such rule systems. Rules are widely deployed today to represent and automate e-business policies and workflows, for example. Practical advances in such inter-operability would offer the promise to greatly facilitate program-to-program knowledge sharing and integration, and thereby to stimulate a global virtuous circle of growing value creation in e-business. In short, we seek to realize rule-based business intelligence on the Semantic Web.

In this paper, we describe the design of SweetJess, our new system for inter-operability of rules between RuleML and Jess. RuleML [17] is an emerging industry standard for XML rules that we (first author) co-lead, being pursued in informal cooperation with the World Wide Web Consortium (W3C) [22] and the DARPA Agent Markup Language (DAML) Program [4]. Rules indeed are part of the announced mission of the W3C's Semantic Web Activity [18]. Jess [11], acronym for "Java

Expert System Shell", is a popular rule system that is free for academic use and whose source code is relatively easily available.

SweetJess is part of our (first author's) larger toolset system SWEET, acronym for "Semantic WEb Enabling Technology". SWEET also includes SweetRules [10], a system for RuleML inferencing and translation, and SweetDeal [8] [9] [15], an approach to rule-based contracting that builds upon SweetRules and RuleML e.g., to represent deals about Web services. Our previous SweetRules prototype was the first to implement SCLP RuleML inferencing and also was the first to implement translation of (SCLP) RuleML to and from multiple heterogeneous rule systems.

"SCLP" stands for the Situated Courteous Logic Programs knowledge representation. The SCLP case of RuleML is expressively powerful. The Courteous feature/extension enables prioritized conflict handling and (a limited form of) classical negation. The Situated feature/extension enables procedural attachments for sensing (testing rule antecedents) and effecting (performing actions triggered by conclusions).

SweetRules enables bi-directional translation from SCLP RuleML to: XSB, a Prolog rule system [23]; the IBM CommonRules rule engine, a forward SCLP system [5]; Knowledge Interchange Format (KIF) [12], an earlier version of the Common Logic [6] emerging industry standard for knowledge interchange in classical logic; and Smodels, a forward logic-program rule engine. SweetJess aims to complement and extend SweetRules by providing additional capabilities including translation to Jess.

The first new contribution of the SweetJess approach is a fundamental mapping. We give a new, implemented translation from a broad but restricted case of SCLP RuleML into Jess rules, and an inverse translation from a broad but further restricted case of Jess rules into SCLP RuleML. The translation is intended to preserve semantic equivalence. Semantic equivalence means that, for a given rulebase, the same conclusions are entailed, and the same side-effectful actions (triggered by conclusions) are performed when the rulebase is executed (i.e., when the rules are "run"). The translation often preserves semantic equivalence; in current work, we are developing formal guarantees for the equivalence, including necessary expressive restrictions in each direction.

The set of expressive restrictions for the translation from SCLP RuleML into Jess includes several that are imposed by the expressive limitations of Jess; notably, these include:

1. Datalog, i.e., no logical functions;
2. safeness/range-restrictedness of rule heads, i.e., every logical variable in the head must appear in (and thus be bound by) the rule body;
3. "all-bound sensors", i.e., sensor attached procedures require all their arguments to be bound (fully instantiated, i.e., ground) when the sensor procedure is invoked; and
4. safe negation, i.e., variables in negated body literals must appear in (and thus be bound by) positive body literals (NB: negation here means negation-as-failure; and
5. dynamically stratifiable negation, i.e., the LP's model under the Well-Founded Semantics [20] must not need to assign any literals the truth value *undefined*.

In addition, there are some other limitations of Jess with regard to negation-as-failure that we are investigating in current work. Finally, Jess also lacks some naming capabilities (especially, for facts and rulebases), as compared to (SCLP) RuleML; however, these are less fundamental expressively than the above restrictions.

The above restrictions (1.), (2.), (4.), and (5.) are often found in rule-based systems — especially ones that do forward-direction inferencing. Restriction (3.), all-bound sensors, is more unusual and is practically rather limiting — it restricts a sensor attached procedure when queried to answer with simply a boolean, as opposed to a set of bindings. Overall, these restrictions underline some dimensions of the powerful expressive generality of SCLP as a KR. We will discuss the translation's expressive restrictions in more detail later.

The second new contribution of the SweetJess approach builds upon the translation. We give a new, implemented architecture to perform (a broad case of) SCLP RuleML inferencing using the Jess rule engine. Our approach translates from SCLP RuleML rules into Jess rules, runs the Jess rule engine to generate conclusions (and actions), and then translates the concluded Jess facts back into SCLP RuleML.

Translating the Courteous feature of SCLP RuleML, i.e., its prioritized conflict handling and classical-negation aspects, is a particular hurdle, since Jess essentially lacks the ability to directly

express these aspects. (Jess does include a quite limited kind of inferencing-control-agenda prioritization — "salience" — which is a discouraged mechanism.) Our approach is able to surmount this hurdle however, by utilizing a Courteous Compiler component. The Courteous Compiler "compiles away" the courteous aspect of an input rulebase, transforming it into a semantically equivalent rulebase that does not contain the Courteous expressive features (priorities and mutual exclusion integrity constraints), but rather that only employs negation-as-failure (NAF). The IBM CommonRules library provides a Courteous Compiler, for example.

The third new contribution of the SweetJess approach is to enable bi-directional implemented inter-operability, via RuleML as an interlingua, between Jess and multiple other heterogeneous rule systems, including Prologs and relational database systems for which translation to RuleML has already been shown, and for which there are existing translation tools (e.g., in SweetRules and our other earlier work [8]). In particular, as we discussed earlier, SweetRules already enables bi-directional translation from SCLP RuleML to: XSB; Smodels; IBM CommonRules; and KIF. The overall approach to such translation was first given by us in [8]. The RuleML website lists additional translation tools as well. For a given rule system such as Jess, the software engineering effort of specification, design and implementation of translation to multiple other rule systems is greatly eased by use of a single intermediate interlingua, i.e., the emerging RuleML standard.

Jess is a representative member of one group of currently commercially important (CCI) rule systems: namely, production rule systems descended from OPS5 [3], which in turn are closely related to event-condition-action (ECA) rule systems [19]. These systems primarily employ forward chaining (rather than backward), and their applications heavily rely on their capabilities for procedural attachments. This group is sometimes called "reactive" for short; often, rules are run in response to the arrival of knowledge-base updates consisting of facts (or "events"). Another quite distinct group of CCI rule systems is comprised of Prolog systems [2], together with SQL-type relational database systems (RDB) [19]. The core of SQL RDB's — relational algebra and Datalog — is well-known theoretically to be very closely related to pure Prolog. Systems in this second group (sometimes called "derivational") primarily employ backward chaining (rather than forward), i.e., query-answering.

The fourth new contribution of the SweetJess approach is a bridging across heterogeneous families of CCI rule systems. Our translation and tool are each the first, to our knowledge, to enable inter-operability between a Prolog and any "production"/"reactive" rule system descended from OPS5 heritage. More generally, our translation may be the first to go for a broad expressive case between the two groups (production/reactive vs. Prolog/SQL derivational).

The fifth new contribution of our translation effort is to compare the expressive capabilities of each rule system and its underlying fundamental knowledge representation, and in particular to bring out several limitations of Jess relative to SCLP RuleML. We discussed the most important of these above.

In continuing the overall SweetRules approach by laying these new foundations for inter-operability, the SweetJess approach thereby moves a discernible step closer to the Semantic Web's vision of wide knowledge sharing and integration among intelligent applications, e.g., where rules are already often deployed for e-business policies and workflow, and SQL RDB's are ubiquitous.

The prototype implementation of SweetJess is and publicly available free on the Web[4]. It is implemented in Java and makes use of tools for XML, RDF [16] and RuleML.

The remainder of this paper is organized as follows. Section 2 provides background: we review RuleML, Situated LP, and Courteous LP. In section 3, we review Jess Rules and begin the analysis and reformulation that underlies our translation mappings. In section 4, we describe SweetJess' architecture to perform (a broad case of) SCLP RuleML inferencing using the Jess rule engine. In section 5, we come to the heart of the matter: we describe how to translate rules from SCLP RuleML to Jess. In section 6, we describe how to translate back from Jess to SCLP RuleML. In section 7, we wind up with some discussion, including additional directions for future work.

[4] http://daml.umbc.edu/sweetjess

2 Background: RuleML, Situated LP, Courteous LP

2.1 RuleML and LP

RuleML [17] is the leading emerging industry standard for XML-based inter-operable rules, i.e., Semantic Web rules. It is being developed by a coalition which includes participants from several dozen institutions, both academic and industrial. We (first author) co-lead it. The specification of RuleML includes an XML markup syntax, together with a formal semantics based on the knowledge representation (KR) of Logic Programs (LP)[5]. In adddition, there already exist early draft specifications of an abstract syntax, an alternative RDF syntax, and an OWL syntax. RuleML is intended primarily to support inter-operability among currently commercially important kinds (CCI) of rule systems. These CCI rule systems fall primarily into four major families today: relational database management systems (SQL), Prolog, OPS5-heritage production rules, and Event-Condition-Action (ECA) systems. The RuleML language actually provides a range of options for the fundamental KR expressiveness to be used in inter-operating between a group of two or more rule systems. Formally, RuleML organizes these options into a set of "sub-languages". Each sub-language provides a different combination of expressive features / expressive restrictions, and has associated syntax. The set of sub-languages is thus hierarchical, in that a given sub-language may provide a superset of (i.e., subsume) the expressiveness of a particular other sub-language.

In the LP literature, the most studied LP expressive class is what we will call "Ordinary" LP (OLP). OLP extends Horn LP by permitting body literals to be negated; its form of negation is negation-as-failure (NAF). Courteous LP subsumes OLP. OLP is also sometimes called "normal logic programs" (or "general [sic] logic programs" as in [1]). OLP is roughly pure Prolog without built-ins, but not limited to backward direction of inferencing.

Currently defined RuleML sub-languages also include:

1. Datalog Horn LP (no negation, no logical functions)
2. Horn LP (adds logical functions (of non-zero arity); no negation; subsumes Datalog Horn)
3. Situated Courteous LP (adds procedural attachments for sensing/effecting, and prioritized conflict handling; subsumes OLP and Horn)

and several others that provide combinations of various additional syntactic or expressive features/restrictions.

Note that Ordinary LP and Horn LP are "*pure-belief*" KR's — i.e., they lack procedural attachments. Most KR theory, generally, treats pure-belief formalisms.

RuleML's sub-languages differ from the formulation of LP expressive classes in the KR literature primarily in that RuleML provides a standardized set of mechanisms for syntax and for Webizing the KR. Besides providing XML[6] syntactic schemas, an important aspect of these mechanisms is to provide additional capabilities for naming, notably to permit predicates, individuals, and logical functions ("constructors") to be URI's, and to permit rule names and rulebase names to be part of the KR language. These naming capabilities facilitate highly distributed knowledge bases and inferencing for rules and their use of ontologies and databases — in short, they largely equip rule KR for the Semantic Web. In particular, permitting a predicate name to be a URI that refers to an OWL class or property enables RuleML rules to be "on top of" OWL ontologies (e.g., see [9] for the first detailed application scenario that used this capability).

RuleML has been evolving since its first version XML DTD's (V0.7) were released in early 2001. The current version (V0.8) includes additional extensions in expressive and syntactic features, and a few moderate revisions in the syntax. Our SweetJess design and implementation effort was started while V0.8 was still in development, and thus used the same version of the SCLP RuleML XML DTD as the first (still current) version of SweetRules [10]. This DTD (known as "SCLP dtd-v13") differs in a few minor ways with RuleML V0.8. In current work, we are updating the implementation to achieve full compliance with the RuleML V0.8 SCLP DTD[7].

The current SweetJess translation mappings and implementation incorporates all the features of the RuleML V0.8 SCLP DTD with one important exception: the object-oriented argument collections feature ("roli's"), which provides additional syntactic convenience (rather than increasing

[5] see [1] for a helpful review of LP KR

[6] and early drafts of RDF and OWL

[7] http://www.ruleml.org/dtd/0.8/ruleml-sclp-monolith.dtd

fundamental expressive power from a logical KR standpoint). The collection of arguments in a logical atom is thus, for now, simply an ordered tuple (in the current SweetJess translation mappings and implementation). Such an atom when ground is called an "ordered fact" in Jess. In current work, we are designing an extension of the SweetJess translation mappings and implementation to support the object-oriented argument collections feature as well.

Next, we give examples of a Fact and a Rule in RuleML V0.8 syntax.

Example 1. **Premium Customer Fact, in RuleML** A RuleML fact ("fact" element in the XML syntax) expresses a ground atom. It is a kind of statement. E.g.:

```
/* Fact: ''Allan is a premium customer.'' */
        premiumCustomer(Allan)
```

In RuleML syntax:

```
<fact>
   <_head>
     <atom>
         <_opr>
            <rel>premiumCustomer</rel>
        </_opr>
        <ind>Allan</ind>
     </atom>
  </_head>
</fact>
```

Example 2. **Discounting Rule, in RuleML** A RuleML implication rule ("imp" element in the XML syntax) expresses an if-then rule (a.k.a. a "clause" in the LP KR literature). It is a kind of statement. An implication rule is a pure-belief rule that does not directly specify any procedural attachments for sensing or effecting. E.g.,

```
/* Rule: ''If a customer is a premium customer then give him a 10% discount.''
        if premiumCustomer(?customer)
        then giveDiscount(percent10, ?customer)
```

For ease of human-readability, in this paper we often give our LP and RuleML examples (e.g., the ones above) in the Prolog-like "SCLPfile" syntax of IBM CommonRules V3.0 [5], which maps straightforwardly to RuleML. ";" ends a rule statement. The prefix "?" indicates a logical variable. "/* ... */" encloses a comment. "//" prefixes a comment line. "< ... >" encloses a rule label (name). Rule labels identify rules for editing and prioritized conflict handling, for example to facilitate the modular modification of contract provisions.

In RuleML syntax:

```
<imp>
   <_head>
      <atom>
         <_opr>
            <rel>giveDiscount</rel>
         </_opr>
         <ind>percent10</ind>
         <var>Customer</var>
      </atom>
   </_head>
   <_body>
      <atom>
         <_opr>
            <rel>premiumCustomer</rel>
         </_opr>
         <var>customer</var>
      </atom>
   </_body>
</imp>
```

For sake of brevity, especially in human authoring and reading, the XML syntax for RuleML uses terse/abbreviated names for the elements (and attributes). "rel" means relation, i.e., predicate. "ind" means individual (object constant). "var" means logical variable. "body" and "head" mean the antecedent (a.k.a. "if" part) and the consequent (a.k.a. "then" part) of a rule, respectively. "opr" stands for relational operator.

Note that a fact is similar syntactically to an (implication) rule that lacks a body. Conceptually, an empty body is viewed as logically True, as is usual in LP and classical logic.

Terminology: From a LP KR viewpoint, a fact is just a special case of a rule. A RuleML rulebase consisting of implication and fact statements is thus often simply called a "rulebase" or "ruleset". More generally, however, the Situated and Courteous features of SCLP extend the LP KR with three additional kinds of statements: sensor, effector, and mutex; we will be discussing those in more detail later. A collection of such SCLP statements (all five kinds) constitutes a RuleML rulebase in the more general sense.

2.2 Situated Logic Programs

The Situated extension of (Courteous or Ordinary) Logic Programs allows actions and queries to be performed by procedural attachments. SLP uses effector and sensor statements to specify these, as in the following example.

Example 3. **Order Cancellation with Notification Action**

```
   /* rule: should notify customer if order cancellation request was received in time to be
accepted */
       <rule_526> if deadlineToCancel(order4215, ?Day)
                        and receivedBefore(cancelRequest4216, ?Day)
                    then shouldInformCustomer(cancelRequest4216, accepted);
   /* effector statement: associated with the predicate shouldInformCustomer, the ack method
performs a notification action */
       Effector: shouldInformCustomer /* the predicate */
       Class: orderMgmt.Request.mods
       Method: ack
       Path: ‘‘edu.cs.umbc.SLP.examples.orderMgmt.aprocs’’;
   /* sensor statement: associated with the predicate receivedBefore, the earlierReceiptDate
method queries an external order management system */
       Sensor: receivedBefore /* the predicate */
       Class: orderMgmt.Request
       Method: earlierReceiptDate
       Path: ‘‘edu.cs.umbc.SLP.examples.orderMgmt.aprocs’’;
       BindingRequirement: (BOUND,BOUND)
```

The effector statements in a SLP each associate a pure-belief predicate, e.g., shouldInformCustomer, with an external attached procedure, e.g., orderMgmt.request.mods.ack (here, a Java method). During rule inferencing (more precisely, during rule execution), when a conclusion is drawn about the predicate, e.g.,"shouldInformCustomer(cancelRequest4216, accepted)" if the rule above was fired successfully, then the external procedure is invoked as a side-effectful action, e.g., the method "ack" is called with its parameters instantiated to "(request1049, accepted)". "External" here means external to the inferencing engine itself. An (external) attached procedure is also called an "*aproc*" for short.

The sensor statements in a SLP each associate a pure-belief predicate, e.g., receivedBefore, with an external attached procedure, e.g., orderMgmt.request.earlierReceiptDate (again, here the aproc is a Java method). During rule inferencing/execution, when a rule antecedent condition (i.e., a literal in the rule's "if" part) is tested, e.g., "receivedBefore(cancelRequest4216,?Day)" in the rule above, then the external procedure is queried to provide information about that condition's truth. More precisely, the aproc is queried for its answer bindings since the condition may contain logical variables. A sensor aproc may require that some or all of its arguments must be bound (i.e., fully instantiated) at the time that aproc is invoked. A sensor statement thus includes a binding pattern that specifies such requirements. Such binding requirements are quite common in practice.

For example, consider an external procedure myCompany.BluePages.getPhoneNumber, provided by a company phone directory application, that has two arguments, where the first is a

person name and the second is a phone number. It has an associated binding pattern that requires the first argument to be bound but permits the second argument to be unbound. When invoked with the first argument bound to "Fred.Green" and the second argument a free variable ("?X"), it returns the binding "617-555-9876" for that variable. In the example above, the sensor aproc orderMgmt.Request.earlierReceiptDate requires both of its arguments to be bound when it is invoked. Some sensor statements, e.g., for the predicate lessThanOrEqual, correspond to what in Prolog (or many other commercial rule systems) are "built-ins", utility procedures provided as a standard package with the rule system rather than provided by a particular individual user/application.

Terminology: a predicate which has one or more associated sensor (effector) statements is called a "sensor predicate" ("effector predicate"). A literal in a sensor (effector) predicate is called a "sensor literal" ("effector literal"). Overall, we call the process of drawing conclusions and performing related effector and sensor invocations in SLP: "*situated inferencing*".

Sensing about a given predicate p occurs in addition to any facts derivable from rules (or facts) whose heads are p atoms. Also, a given predicate p may appear in multiple sensor statements, i.e., p may have multiple sensor aproc's. The results from querying all of these sensor aproc's are combined (i.e., union'd) when a rules body's sensor literal in p is tested.

Likewise, a given predicate p may appear in multiple effector statements, i.e., p may have multiple effector aproc's. When a conclusion is drawn about p, each of these effector aproc's is invoked.

2.3 Courteous Logic Programs: Review

Courteous Logic Programs (CLP) is an expressive super-class of Ordinary Logic Programs (OLP). The Courteous expressive extension enables prioritized conflict handling and also a limited form of classical negation. Next, we give an example of a CLP, having 2 rules, 1 fact, and 1 mutex.

Example 4. **Prioritized Discounting Rules**

```
/* if the Customer has a Loyal Spending History then give him a 5% Discount */
 <steadySpender>
      IF shopper(?Cust) and spendingHistory(?Cust, loyal)
      THEN giveDiscount(percent5, ?Cust);
  /* if the Customer was Slow to Pay last year then give him a 0% (NO) Discount */
 <slowPayer>
      IF slowToPay(?Cust, last1year)
      THEN giveDiscount(percent0, ?Cust);
  /* prioritization fact: SlowPayer is higher priority than SteadySpender */
      overrides(slowPayer, steadySpender);
  /* the amount of the Discount given to a customer is Unique */
      MUTEX giveDiscount(?X, ?Cust) and giveDiscount(?Y, ?Cust)
      GIVEN notEquals(?X, ?Y) ;
```

Each rule has an optional rule label. This is used as a handle for specifying prioritization information. Each label is a logical term, e.g., an individual constant. The "overrides" predicate is used to specify prioritization. "overrides(lab1,lab2)" means that any rule having label "lab1" is higher priority than any other rule having label "lab2". "overrides" is syntactically reserved, but otherwise is treated as an ordinary predicate. In particular, "overrides" can itself be the subject of inferencing. The scope of what is conflict is specified by "*mutex*" statements. A mutex specifies a (conditional) pair-wise mutual exclusion between two literals; these are called the "*opposer*" literals of the mutex. The mutex statement also includes a condition, called its "*given*" part. The mutex given part is similar to the body of a rule; it may be empty (i.e., $True$). E.g., the mutex above specifies that it is a contradiction to conclude two different values of the percentage discount for the same customer; i.e., `giveDiscount` is a (partial-)functional predicate. The semantics of Courteous LP enforces consistency of the conclusions wrt each mutex.

Any literal may be classically negated; however, (C)LP as a KR only supports a quite limited form of classical negation (a.k.a. "strong" negation). There is an *implicit* mutex between p and classical-negation-of-p, for each p, where p is a predicate, ground atom, or atom. These implicit mutex's are called "*classical-negation*" mutex's.

The semantics of Courteous LP ensures overall consistency of the conclusion set, including consistency between the two concepts of negation (classical negation of p entails NAF of p, but not vice versa).

Combining Courteous + Situated → SCLP: The Courteous expressive extension can be combined with the Situated expressive extension, to form Situated Courteous LP. Currently in the theory of Situated Courteous LP, however, there is an *expressive restriction* on this combination: the sensor predicates must be conflict-free. More precisely, the restriction on the SCLP rulebase is that: no mutex opposer literal may be a sensor literal. This includes the implicit classical-negation mutex's, thus no sensor literal may be classically-negated. We call this restriction "*conflict-free sensing*" or "*monotonic sensor predicates*", for short. In current work, we are generalizing the theory of SCLP to remove this restriction.

3 Jess Rules: Overview, Analysis, and Reformulation

3.1 Jess Facts

Conceptually, what Jess calls a "fact" is very similar to the concept of a fact in RuleML; it expresses a ground atom. Jess provides some machinery for defining facts and acquiring them from surrounding Java context, with which we will not need to concern ourselves in this paper. We will concern ourselves simply with the basic concept of a Jess fact — specifically, what Jess calls an "ordered" fact, i.e., one where the fact's arguments constitute an ordered tuple. Jess also provides as a syntactic enhancement the concept of an "unordered" fact, which uses slot names (rather than sequence) to define arguments within a fact (or atomic pattern); this is quite similar to the object-oriented argument collections feature of RuleML V0.8, which we mentioned earlier. The current SweetJess translation mappings and implementation just deal with Jess "ordered" facts, however.

Jess uses a Lisp-like syntax, generally. A Jess fact statement has the following kind of form (we can view this roughly as a generic template):

```
(assert (predicateName constant1 constant2 ...constantN) )
```

The statement begins with the "`assert`" keyword. This is followed by an expression syntactically similar to an LP ground atom: a predicate followed by an ordered collection of individual constants. Different predicates have different arities; we indicated this in the form above by writing "N" as the arity. Actually, in Jess, "`assert`" is a system procedure (what Jess calls a "function"); when executed it puts the fact into the currently active stored knowledge base of the Jess inferencing engine. Note that, in the tradition of OPS5-heritage production rule systems, Jess does not conceptually view a Jess fact as a special case of a Jess rule. However, in our translation mapping, we will essentially view a Jess fact as a special case of a *LP* rule, from the viewpoint of KR theory.

Jess lacks the semantic equivalent of logical functions, i.e., constructors, that have non-zero arity. It thus also lacks the semantic equivalent of complex terms formed using constructors. This is known in LP and classical logic KR as the *Datalog* restriction. The closest thing in Jess to a non-zero-arity constructor is a JessMethod procedure (see next sub-section), but that is always evaluated on its arguments; the semantics of a non-zero-arity constructor, however, essentially correspond to not evaluating it.

3.2 Jess Rules

A Jess rule has the following kind of syntactic form (we can view this very roughly as a generic template):

```
(defrule ruleName
(predicate1 constant1 ?boundVariable1)
(test (jMethod1 constant2 ?boundVariable1))
=>
(jMethod2 (symbol3 ?boundVariable1)) )
```

A Jess rule definition begins with the "`defrule`" keyword followed by a rule name, and has two further parts, an "if" part on its left hand side (LHS) and a "then" part on its right hand side (RHS), separated by the "`=>`" symbol which roughly means implication. The LHS of a Jess rule is a "*pattern*" in Jess terminology. A *basic* "pattern" matches facts and corresponds to an atom in

a LP rule body. E.g., the second line in the example form above is a basic pattern. This atom may include logical variables, not just individual constants, as arguments. More complex patterns can be formed in several ways. The LHS may consist of a (top-level) list of basic patterns; these are interpreted (implicitly) as a conjunction.

The basic form of a Jess rule's RHS is an "*action*" in Jess terminology. An action is simply a call to one of Jess' Java procedures. We will call these procedures "JessMethods". (Jess calls them "functions", but we wish to reserve "function" for logical functions cf. LP and classical logic KR terminology.) Syntactically, a JessMethod call is simply a Lisp-like list whose first member is the name of a JessMethod, followed by arguments. Each such argument may be a constant, logical variable, or another JessMethod procedure call expression. E.g., the last line in the template example above is an action. "Jess comes with a large number of built-in [JessMethods] that do everything from math, program control and string manipulations, to giving you access to Java APIs.", says the User Manual[8]. In addition, a user may define their own additional JessMethods, which are essentially general Java procedures (methods); in practice, this capability is often used heavily. In general, an action may essentially arbitrarily modify program state, and is a way to do pretty much anything you can do in Java.

From the standpoint of KR and for designing our translation mappings, however, one JessMethod is particularly germane: "`assert`". A common kind of action in Jess rules is to `assert` a fact. In this case, the Jess rule RHS corresponds to the head atom of a (pure-belief) implication rule in LP KR, and thus in RuleML. When the Jess rule inferencing engine runs, if the rule LHS is satisfied ("matches" in Jess terminology), i.e., if the rule fires (with variable bindings supplied as in the usual manner for rules), then this RHS fact is added to the working fact set portion of Jess' knowledge base. From the standpoint of LP KR, this corresponds to drawing a conclusion. More generally, the RHS may consist of a (top-level) list of actions. If these are `assert`'s, the list is interpreted (implicitly) as a conjunction of its member actions.

The result of the RHS when a rule fires thus may be either (1) to draw a conclusion fact (or several facts), or (2) to perform some (general) procedural action(s) that may be — and typically, are – side-effectful. From the standpoint of Situated LP KR, expressively, case (2) corresponds essentially to generating an effector call to a JessMethod. (A complex RHS can be rewritten as a call to a single JessMethod.)

In addition to basic patterns that correspond to LP atoms, there is one other fundamental kind of (pattern) expression can appear in the LHS of a Jess rule: a "TEST Conditional Element" in Jess terminology, called "TestCE" for short.[9] [10] A TestCE is constructed syntactically using the reserved keyword "`test`" followed by a JessMethod call expression (as in the third line of the template example above). When the rule runs, the TestCE tests whether that JessMethod call expression (when supplied with variable bindings by matching the rest of the LHS) evaluates to True (vs. False). From the standpoint of Situated LP KR, this essentially corresponds to a sensor call. When the JessMethod call is invoked, all the arguments of the JessMethod must be fully bound. This is significantly less (expressively) general than the concept of a sensor call in Situated LP, where some or all of the arguments may be (or contain) free variables. We call this the *all-bound sensors* expressive restriction.

Jess also has explicit logical connectives. The first is "`not`", which is negation-as-failure. "`(not` BP`)`, where BP is a basic pattern, is similar to a negation-as-failure literal in LP. More complex patterns can be built up by explicit use of logical connectives; these include `not`, `and`, `or`, `exists`, and can be nested. This provides enhanced expressiveness in a direction very similar to (but somewhat less general than) "*Lloyd-Topor*" LP. "Lloyd-Topor" LP is the extension of OLP to use the Lloyd-Topor transformation [13]. The Lloyd-Topor transformation does not increase the fundamental expressiveness of OLP, however; it reduces its more expressive version of LP into OLP. In that sense, it can be viewed as syntactic sugar.

[8] Jess V6.1, section 2.2

[9] In addition, Jess permits "predicate constraints" and "return value constraints" to be associated with LHS variables, but these do not provide any extra essential expressiveness; they are reducible to TestCE's plus the other permitted LHS pattern constructs.

[10] Note that a LHS rule pattern may not contain a top-level JessMethod appearance, only `test`.

Jess requires that all logical variables appearing in the RHS be bound by matching in the LHS. I.e., all RHS variables must appear in the LHS. This is known in LP and classical logic KR as *range-restrictedness* and as the *safe-head* expressive restriction. Jess also requires that: all logical variables appearing in a basic pattern (or more complex expression) that is negated (by "`not`"), must be bound by matching (on non-negated patterns) in the rest of the LHS. [11] This is known in LP and classical logic KR as the *safe negation* expressive restriction. Safe-head and safe-negation are frequent restrictions in practical rule-based inferencing systems, and are especially common in forward-direction inferencing systems.

Jess Rule Engine: Above, we have described a Jess rule and a Jess fact, taken one a time. Overall, the KR semantics of a set of Jess rules and facts must be related to what the Jess engine does — what conclusions are drawn and what procedural actions are performed — not just to what is the conceptual intention of a rule or fact. From a KR viewpoint, the Jess engine is similar to the engines of several other OPS5-heritage production rule systems. It uses the Rete (Latin for "net") algorithm [7] for efficiency in "pattern" matching, especially to handle updates to its working set of facts.

In current work, we are developing more formal theory to characterize the production-rule Rete engine algorithm in terms of the LP KR.

4 SweetJess' Overall Architecture for Inferencing and Translation

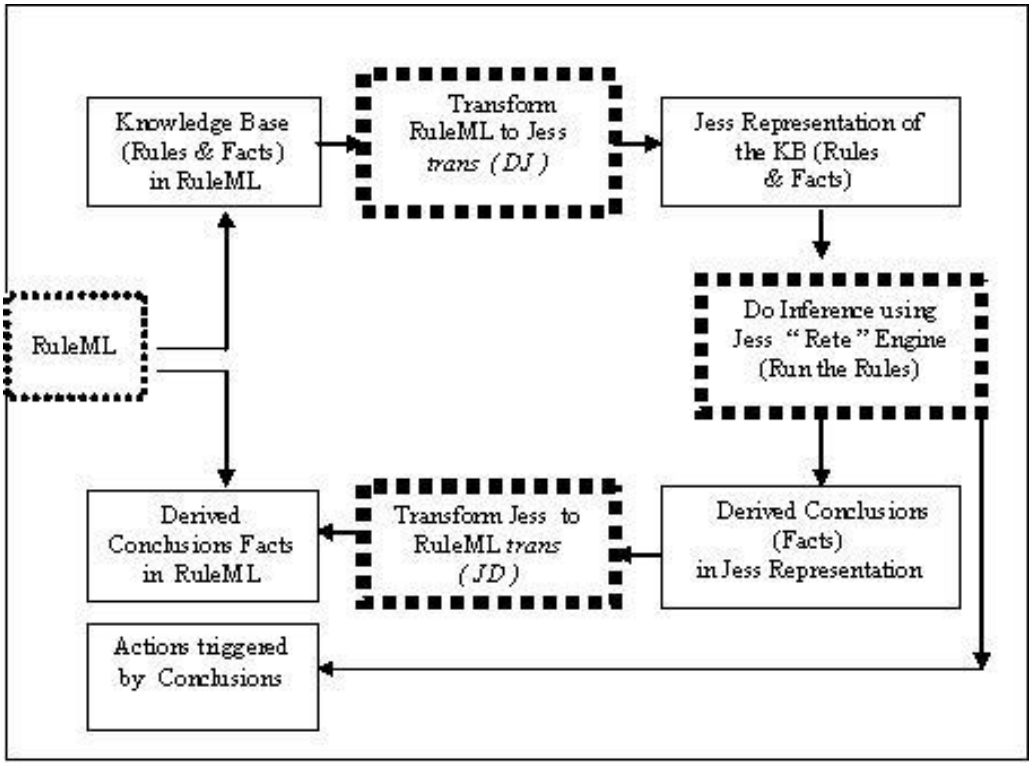

Fig. 1. SweetJess Architecture for RuleML Inferencing via Jess

The bi-directional translation between RuleML and Jess has several potential uses, as discussed in section 1. By way of motivation for the details in the rest of this paper, however, we largely

[11] Jess also allows local existentially-quantified variables within a negated expression, but this is expressively inessential.

focus on one particular use: to perform RuleML inferencing via Jess. Figure 1 shows SweetJess' architecture for this. By "RuleML inferencing", we mean (situated) inferencing from a premise RuleML rulebase ("Knowledge Base of rules and facts" in the Figure) to derive RuleML conclusions and related procedural actions that are triggered from those conclusions via procedural attachments ("effectors" in Situated LP, described in section 2.2). By "via Jess", we mean that Jess is used as an engine to "run" the rules.

To perform such inferencing, SweetJess first translates a premise RuleML rulebase into a set of Jess rules and facts (and JessMethod definitions). This transformation is called trans[RJ]. Inferencing is then performed using Jess' rule inference engine, i.e., the rules are "run". This generates a set of derived conclusions (facts) in the Jess representation. Then these facts are transformed by a SweetJess translator component into a set of RuleML facts. This inverse direction transformation is called trans[JR]. The result is a set of RuleML conclusion facts entailed by the original RuleML premise rulebase.

When the rules are run in the Jess engine, a set of actions, triggered by conclusions, is also performed. These actions are invocations of attached procedures, i.e., side-effectful calls to Java methods. These actions are those sanctioned by the Situated ("effecting") aspect of the semantics of the premise RuleML rulebase. The transformations trans[RJ] and trans[JR] impose some expressive restrictions on their input, which we will describe later. For the results of inferencing, only facts need be translated via trans[JR] from Jess to RuleML. Our trans[JR] translation, more generally, actually handles (as input) Jess rules too, not just Jess facts. The current implementation of trans[JR], however, restricts these input Jess rules to be pure-belief rules, i.e., without JessMethods (other than `assert`), and to be in similar syntactic form to what trans[RJ] produces. In current work, we have been extending the design of trans[JR] to relax these restrictions.

5 Transforming RuleML To Jess: trans[RJ]

There are several major challenging aspects of designing the translation mappings between RuleML and Jess. One aspect is to map conceptually between the terminologies and, more deeply, the concepts of the two different rule languages. We described much of that conceptual mapping earlier in sections 3 and 4. A second aspect is to handle procedural attachments for tests/sensing and actions/effecting. A third aspect is to handle prioritized conflict handling. A fourth aspect is to identify the appropriate expressive restrictions in doing all of the above.

Next, we describe in more detail the various aspects of trans[RJ]. The topics of the following subsections are sequenced roughly in the sequence of increasing expressiveness: from facts to Horn LP to OLP to Situated OLP to Situated Courteous LP. As we go, we sometimes need to distinguish the translation mapping (which is at the level of design) from the current implementation of SweetJess.

5.1 Overall Input and Output

The transformation trans[RJ] takes as input a RuleML rulebase — an XML document which we will call a .rml file. The output of this transformation is a Jess knowledge base — i.e., a .jess batch file. This batch file contains facts and rules which can be directly fed to a Jess Rete engine. The batch file begins with the Jess system command "`(reset)`", which when executed clears out the knowledge base and resets the engine. The batch file ends with the Jess system command "`(run)`", which when executed starts up inferencing. The current version of Jess is V6.1. The current version of RuleML is V0.8. As we discussed earlier in section 1, the current SweetJess implementation still uses a slightly earlier version of the XML DTD for SCLP RuleML.

5.2 Fact

A basic RuleML fact — an atom whose arguments are an ordered tuple of individuals and/or variables — translates straightforwardly into a Jess fact.

Example 5. **Premium Customer Fact, in Jess:** The equivalent Jess fact corresponding to the RuleML fact in Example 1 is as follows:

```
(assert (premiumCustomer Allan) )
```

URI constant names feature: An important Webizing feature of RuleML is "URI constant names": predicate or individual (or other) constant names can be be URI's. A very basic way to translate these in trans[RJ] is simply to treat them as strings; however, then they are not recognized as being URI's by Jess. Translating them round-trip from RuleML to Jess back to RuleML in such a way as to preserve the ability to recognize URI-ness upon the return to RuleML, is relatively straightforward to support in trans[RJ] and trans[JR], e.g., via a prefixing/encoding convention. Our translation mapping design thus includes URI constant names, but the current implementation does not yet support this feature.

Rule naming issues: In RuleML, a fact has an optional rule label (name). A fact in Jess has a unique Fact-Id which is generated by the system upon loading, and is only accessible to the system rather than being explicit in its Jess representation. The RuleML fact's rule label, if present, is thus lost by the transformation. Thus formally, there is an expressive restriction on the transformation: "*no fact labels*". (An alternative design to relax this restriction would be to translate a RuleML fact into a Jess rule that has a trivially true body, then the rule label could be preserved. But this would probably have some other disadvantages, e.g., less efficiency.)

Object-oriented argument collections: The current design of the translation mapping also does not yet support the object-oriented argument collections feature of RuleML V0.8. That recently-added feature, as yet experimental in status, provides syntactic and conceptual convenience but does not add fundamental expressiveness from a KR standpoint. We will call this restriction the "*tuple-arguments*" expressive restriction.

Expressive restrictions imposed on the rulebase: Datalog, no fact labels, tuple-arguments.

Expressive features enabled for rulebase: URI constant names.

5.3 Horn Rule

A RuleML Horn LP implication rule (i.e., `imp` statement) is translated into a Jess rule whose LHS corresponds to the RuleML body, and whose RHS corresponds to the RuleML head. Each RuleML body atom is translated into a corresponding basic pattern. The RuleML head atom is translated into an assert of the corresponding basic pattern. The RuleML rule label, if present, is translated into the Jess rule name, else a new rule name is automatically generated.

Example 6. **Discount Rule, in Jess** After transforming the RuleML rule given in Example 2 (sub-section 2.1), the corresponding Jess rule is:

```
(defrule discountRule
   (premiumCustomer ?customer)
   =>
   (assert (giveDiscount percent10 ?customer) ) )
```

A further subtlety arises when translating rulebases that contain sensor statements. The sensor statements modify how a rule mentioning a sensor predicate is translated so as also to generate TestCE patterns in place of basic patterns; see sub-section 5.6 below for details.

Naming Issues: A RuleML rule label (name) is optional and need not be unique. Jess requires a name for every rule, w hich moreover must be unique (if not, the last-loaded with that name blows away any previously-loaded rule with the same name). For the time being, we thus expressively restrict the input RuleML rules' labels not to coincide with each other. We call this the *unique rule labels* expressive restriction.

Expressive restrictions imposed on the rulebase: unique rule labels, safe head. Note also that the Datalog restriction applies to the LP rule labels, thus each rule label must be simply an individual constant.

Expressive features enabled for the rulebase: Horn LP.

5.4 Lloyd-Topor And-Or (LTAO) Expressiveness

The "*Lloyd-Topor And-Or (LTAO)*" permits: (1) conjunction (of literals) to appear in a rule head; and (2) disjunction to appear in a rule body, even nested with conjunction to form AND-OR expressions formed from literals. This is a portion of the Lloyd-Topor transformation: a rule

$a \wedge b \leftarrow c$ can be rewritten as two rules $a \leftarrow c$ and $b \leftarrow c$; and a rule $a \leftarrow (b \vee c)$ can be rewritten as two rules $a \leftarrow b$ and $a \leftarrow c$.

trans[RJ] supports the LTAO expressive feature. LTAO is convenient but expressively inessential. The SCLP RuleML V0.8 sub-language includes LTAO; many practical rule-based systems support it. LTAO can be added (inessentially) to Horn LP or any of its expressive super-classes (e.g., OLP).

An advantage of this LTAO feature is conciseness and naturalness in authoring of rules. The LTAO feature is straightforward to implement.

Expressive features enabled for the rulebase: Lloyd-Topor And-Or (LTAO).

5.5 Negation-as-Failure (NAF)

Negation-as-failure (NAF) in OLP and SCLP RuleML is translated into Jess negation ("`not`"), which also is NAF.

NAF in OLP (and thus in SCLP), and in rule systems generally, is somewhat tricky in the fully general case, both to define semantically and to implement. As is well-known in the OLP literature, NAF can cause semantic trouble by interacting with cyclic dependencies ("recursion") among rules. A full discussion of the subtleties of NAF is beyond the scope/space of this paper. But we do impose the following restriction: *dynamically stratifiable negation*, i.e., the (O)LP's model under the Well-Founded Semantics [20] must not need to assign any literals the truth value *undefined*. (Note that NAF-free is a special case of dynamically stratifiable.)

In addition, there are some other limitations of Jess with regard to negation-as-failure that we are investigating in current work.

Expressive restrictions imposed on the rulebase: safe negation, dynamically stratifiable negation.

Expressive features enabled for the rulebase: Negation-As-Failure (NAF), thus OLP.

5.6 Situated LP Procedural Attachments for Sensing and Effecting

Next, we describe how sensor and effector statements, in the Situated feature of RuleML, are translated.

An effector statement associating a predicate p with an attached procedure q, is translated into two Jess statements. (1) The first is a Jess rule whose LHS is an open atom in the predicate p and whose RHS is a JessMethod that invokes the aproc q. Here, "Open" atom means that all of the atom's arguments are (free) logical variables. (2) The second is a JessMethod definition statement (using the `deffunction` keyword/command), that defines a a new "user-defined" JessMethod that corresponds to the effector aproc q. There are some other relatively straightforward mechanics of how to pass the path, classname, and methodname of a (sensor or effector) aproc to Jess when defining the corresponding JessMethod; some of these details are in evidence in the example below. There are a few different possible lower-level design choices for exactly how to do this. Below, we illustrate and describe how it is done in the current implementation.

Example 7. **Notification Effector**: The effector statement from Example 3 (sub-section 2.2) is translated into the following Jess rule. (Also — not shown here — there is a JessMethod definition statement for the effector aproc `ack`.)

```
(defrule effect_shouldInformCustomer_1
    (shouldInformCustomer ?orderModificationRequest ?status)
    =>
    (effector ack orderMgmt.Request.mods
                (create$ ?orderModificationRequest ?status) ) )
```

The "`effector`" JessMethod in the Jess rule above is actually a *generic* effector procedure (one for the whole rulebase) that takes the particular effector aproc's methodname, classname, and effector-call argument list as its input parameters. This is a rather elegant low-level design approach that exploits the power of Java in Jess. This generic effector procedure also has a corresponding JessMethod definition statement. Similarly, in this approach, there is a generic "`sensor`" JessMethod (one for the whole rulebase).

A sensor statement associating a predicate p with an attached procedure q, is translated more indirectly. Its presence in the input results in modifying the translation of every rule r whose body mentions the predicate p. A sensor atom of the form $p(t)$, within the body of r, is then translated into the expression

`(or` $BP(p,t)$ `(test` $MC(q,t)$`) )`

rather than into simply the basic pattern $BP(p,t)$ which corresponds directly to the sensor atom. Here, t stands for a tuple of arguments, and $MC(q,t)$ is a JessMethod call expression to invoke aproc q on arguments t. That is, a TestCE pattern is also generated to do sensing via the aproc q, and that TestCE pattern is disjoined (i.e., OR'd) with the basic pattern in p. The result when the rule LHS is matched/tested is to invoke/query the sensor as well as to match the basic pattern against the working memory's set of facts. Also, like with effectors, the translation generates a JessMethod definition statement for the sensor aproc q.

More generally, there may be multiple (e.g., two) sensor statements for p, each with a corresponding aproc, e.g., $q1$ and $q2$. In this case, multiple TestCE patterns are disjoined — one per aproc — e.g., the sensor atom $p(t)$ is translated into

`(or` $BP(p,t)$ `(test` $MC(q1,t)$`) (test` $MC(q2,t)$`) )`

Example 8. **Receipt Date Sensor**: In Example 3, the rule together with the sensor statement, is translated into the following Jess rule.

```
(defrule rule_526
    (deadlineToCancel order4215 ?Day)
    (or
       (receivedBefore cancelRequest4216 ?Day)
       (test (sensor earlierReceiptDate orderMgmt.Request
                     (create$ cancelRequest4216 ?Day) ) ) )
    =>
    (assert (shouldInformCustomer cancelRequest4216 accepted) ) )
```

(Also — not shown here — there is a JessMethod definition statement for the sensor aproc `earlierReceiptDate`.)

Expressive restrictions imposed on the rulebase: all-bound sensors (recall section 3).

Expressive features enabled for the rulebase: Situated LP (sensors, effectors), thus Situated OLP.

5.7 Courteous Prioritized Conflict Handling

So far, we have described how to translate for the Situated Ordinary LP (SOLP) expressive class of RuleML (along with the LTAO and URI constant names features). We call this the *SOLP case* of the translation. SCLP RuleML also includes the Courteous expressive feature for prioritized conflict handling (and limited classical negation), which we overviewed in sub-section 2.3. Jess, however, does not support anything like the Courteous feature directly; it lacks the ability (directly) to express mutex's or the kind of prioritized conflict handling that Courteous LP enables.

Use Courteous Compiler: We have found a means for overcoming this incapacity of Jess. It is to compose an additional transformation, called the Courteous Compiler, as a first step before the "basic" SOLP-case translation. As we have shown in previous work [8] [5], the Courteous Compiler transforms a (Situated) Courteous LP into a semantically equivalent (Situated) Ordinary LP. IBM CommonRules and SweetRules make use of a Courteous Compiler component, for example; CommonRules provides one as part of its toolset. [12] For an input SCLP RuleML rulebase that contains Courteous expressive features (notably, mutex's explicit or implicit), we thus refine the SweetJess architecture accordingly: as a first step in trans[RJ], the input RuleML rulebase (.rml) is transformed via a Courteous Compiler (CC) component into a different, but semantically equivalent, RuleML rulebase that no longer contains the Courteous features. This post-CC rulebase is then run through the basic SOLP-case translator for trans[RJ]. The SOLP-case translator handles SOLP (plus LTAO and URI constant names).

[12] The Courteous Compiler step is tractable computationally: worst-case $O(n^3)$ but typically more like $O(k*n)$, where $3 \leq k \leq 50$, in practical experience to date.

Expressive features enabled for the rulebase: Courteous (prioritized conflict handling, limited classical negation), thus SCLP.

Expressive restrictions imposed on the rulebase:
(1) the *post-Courteous-Compiler NAF-related* restrictions (dynamic stratifiability)[13]; and
(2) *conflict-free sensing* (recall sub-section 2.3).
Note also that the Datalog restriction applies to the mutex's.

6 Transforming Jess To RuleML: trans[JR]

Next, we describe trans[JR], the translation from Jess to RuleML. In the rest of this section, the "translation" means trans[JR], unless explicitly indicated otherwise.

The input to trans[JR] is a Jess batch (.jess) file containing facts, rules, and JessMethod definitions, that can be directly fed to the Jess Rete engine in its current version (V6.1). Output of trans[JR] is a RuleML (.rml) file.

To support RuleML inferencing via translation to/from Jess, via our SweetJess architecture, it suffices simply to translate Jess facts — i.e., the conclusions of inferencing by the Jess rule engine — to RuleML.

The translation of facts is straightforward. Jess facts are defined in calls to the JessMethod "`assert`". To transform a Jess fact, trans[JR] just strips off the `assert` to obtain the inner ground-atom-like expression, and generates a RuleML fact that corresponds to that inner expression. Facts in Jess each have a unique Fact Id which is generated by the system upon loading or inferencing. Jess facts do not have an explicit label for identification. In RuleML facts have an optional rule label. For the time being, we define trans[JR] to simply translate this Jess Fact Id into the RuleML fact label.

Our translation mapping extends to much more than facts, however. It is relatively straightforward to invert the OLP case of trans[RJ], i.e., to "round-trip" the results of trans[RJ]; each Jess rule is translated into a RuleML (implication) rule.

The fundamental expressive class covered by trans[JR] thus includes OLP (with the other OLP-relevant expressive features and restrictions that we discussed in the last section). The current prototype implements this case of trans[JR]. However, in the current implementation of trans[JR], the Situated extension is not supported, i.e., the implementation of trans[JR] does not handle TestCE's or (non-`assert`) actions.

In the larger SweetJess effort, we have been investigating how to extend the fundamental expressive class covered by this translation trans[JR] from OLP to *Situated* OLP. Next, we give a sketch of how.

(1.) For each JessMethod *sproc* appearing in some TestCE:
(a.) introduce a new predicate *spred*; and
(b.) generate a sensor statement associating *spred* with *sproc*.
Note this sensor statement also essentially defines/declares *sproc* from RuleML's viewpoint.

(2.) Likewise, for each JessMethod *eproc* appearing in some action (expression):
(a.) introduce a new predicate *epred*; and
(b.) generate an effector statement associating *epred* with *eproc*.
(Similarly, this effector statement also essentially defines/declares *eproc* from RuleML's viewpoint.)

(3.) Translate a TestCE pattern `(`*sproc t*`)` to a sensor atom $spred(t)$, where t stands for the arguments.

(4.) Translate an action `(`*eproc t*`)` to an effector atom $epred(t)$, where t stands for the arguments.

7 Conclusions, Discussion and Future Work

For the main Conclusions, see section 1 "Introduction and Overview", especially the list of novel contributions we gave there. At core, our effort is not particular to RuleML or Jess, but rather

[13] In current work, we are investigating how to characterize syntactic sufficient conditions on the original (S)CLP to guarantee this restriction is met.

between knowledge representations. Its essence is to translate from declarative SCLP to (OPS5-heritage) production rules, and vice versa. This continues the overall approach and vision we first gave in [8].

That the translation between RuleML and Jess imposes some expressive restrictions in each direction is entirely typical when engaged in defining translations between two heterogeneous rule systems (or any other kind of heterogeneous systems) — the translation handles their expressive overlap.

Another contribution of our translation effort is to discover and compare the expressive capabilities of each rule system and its underlying fundamental KR. In particular, we discovered and highlighted some limitations of Jess as compared to SCLP, including about its ability to represent attached procedures. Jess is less expressively powerful than Situated (Courteous) LP, in that sensor arguments must be fully bound, and sensors may only return true or false; whereas in SCLP, sensor arguments may contain variables that are unbound at the time the sensor is called, and sensors may return sets of bindings (or sets of facts, viewed alternatively). Jess also can make use of Courteous prioritized conflict handling since Jess does not provide a comparably powerful or clean way to express prioritized conflict handling. The comparative insights emerging from the translation effort thus show the potential value of SCLP as an expressive enhancement relative to production rule systems.

Our current work includes implementation and testing of the translation mapping and the overall architecture; development of more formal theory/theorems about the semantic equivalencies including about correctness of the translation and about semantics of negation-as-failure; extending to object-oriented argument collections; and integration with SweetRules. In this regard, there may be some additional, relatively minor, expressive restrictions to be added, or other relatively minor modifications needed, to ensure the correctness of the translation. The version of the translation design in this paper is penultimate, rather than finalized, in that sense.

References

1. Baral C. and Gelfond M.,"Logic Programming and Knowledge Representation", J. Logic Programming, 19-20: 73-148
2. Clocksin W.F. and Mellish C.S., Programming in Prolog. Springer-Verlag, 1981
3. T. Cooper and N. Wogrin, Rule-Based Programming with OPS5. Morgan-Kaufmann Pub., 1988
4. DARPA Agent Markup Language Program http://www.daml.org/
5. IBM CommonRules. http://www.alphaworks.ibm.com/
6. Common Logic, a proposed ISO standard. http://cl.tamu.edu/
7. Forgy, Charles L., "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem", *Artificial Intelligence* 19(1), pp. 17-37, 1982.
8. Grosof B.N., Labrou Y., and Chan H.Y., "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML". Proc. 1st ACM Conf. on Electronic Commerce (EC-99), 1999.
9. Grosof, B.N., Poon, T.C., "Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions". Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003), 2003. Earlier version in: Proc. Intl. Wksh. on Rule Markup Languages for Business Rules on the Semantic Web, held at 1st Intl. Semantic Web Conf., 2002.
10. Grosof B.N., "Representing E-Business Rules for Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML". Proc. Wksh. on Information Technology and Systems (WITS '01), 2001. Extended report version available at author's website.
11. Jess. http://herzberg.ca.sandia.gov/jess/.
12. Knowledge Interchange Format. http://logic.stanford.edu/kif and http://www.cs.umbc.edu/kif. Closely related is the new Common Logic effort.
13. John W. Lloyd, *Foundations of Logic Programming, Second, Extended edition*, Springer, Berlin, 1987.
14. Niemela, I. and Simons, P., Smodels (version 1). http://saturn.hut.fi/html/staff/ilkka.html.
15. Reeves D.M., Wellman M.P. and Grosof B.N., "Automated Negotiation From Declarative Contract Descriptions". *Computational Intelligence*, special issue on Agent Technology for Electronic Commerce, Nov. 2002. (Revised and extended from 2001 Autonomous Agents conference paper.)
16. Resource Description Format (RDF) from World Wide Web Consortium. http://www.w3.org.
17. Rule Markup Language Initiative. http://www.ruleml.org and http://www.ebusiness.mit.edu/bgrosof/#RuleML.
18. Semantic Web Activity of the World Wide Web Consortium. http://www.w3.org/2001/sw.
19. Ullman J.D. and Widom J., A First Course in Database Systems. Prentice-Hall, 1997.

20. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf, "The Well-Founded Semantics for General Logic Programs," *Journal of the ACM*, 38:3, pp. 620-650, July 1991. http://www.cs.columbia.edu/~kar/pubsk/wfjacm.ps.
21. Web Services Activity of the World Wide Web Consortium. http://www.w3.org/2002/ws.
22. World Wide Web Consortium. http://www.w3.org.
23. XSB logic programming system. http://xsb.sourceforge.net./ and http://www.sunysb.edu/ sbprolog.
24. XSLT (eXtensible Stylesheet Language Transformations), http://www.w3.org/Style/XSL/.

Extending the SweetDeal Approach for E-Procurement using SweetRules and RuleML

Sumit Bhansali and Benjamin N. Grosof

Massachusetts Institute of Technology,
Sloan School of Management, Cambridge, MA 02139, USA
{bhansali,bgrosof}@mit.edu; http://ebusiness.mit.edu/bgrosof

Abstract. We show the first detailed realistic e-business application scenario that uses and exploits capabilities of the SweetRules V2.1 toolset for e-contracting using the SweetDeal approach. SweetRules is a uniquely powerful integrated set of tools for semantic web rules and ontologies. SweetDeal is a rule-based approach to representation of business contracts that enables software agents to create, evaluate, negotiate and execute contacts with substantial automation and modularity. The scenario that we implement is of electronic procurement of computers, with request-response iterated B2B supply-chain management communications using RuleML as content of the contracting discovery/negotiation messages. In particular, the capabilities newly exploited include: SweetJess or SweetXSB to do inferencing in addition to the option of SweetCR inferencing, SweetOnto to incorporate/merge-in OWL-DLP ontologies, and effectors to launch real-world actions. We identify desirable additional aspects of query and message management to incorporate into RuleML and give the design of experimental extensions to the RuleML schema/model, motivated by those, that include specifically: fact queries and answers to them. We present first scenario of using SCLP RuleML for rebates and financing options, in particular exploiting the courteous prioritized conflict handling feature. We give a new SweetDeal architecture for the business messaging aspect of contracting, in particular exploiting the situated feature to exchange rulesets, that obviates the need to write new (non-rule-based) agents as in the previous SweetDeal V1 prototype. We finally analyze how the above techniques, and SweetDeal, RuleML and SweetRules overall, can combine powerfully with other e-business technologies such as RosettaNet and ebXML.

1 Introduction

In this paper, we describe in detail a practical electronic contracting scenario that uses RuleML[1], the Situated Courteous Logic Programs (SCLP) knowledge representation [6], and the SweetRules V2.1 semantic web rules toolset [2] together to show how a real-world business application such as electronic procurement can be supported with semantic web technologies including also OWL [3]. The electronic procurement application was chosen not only because of its wide applicability in e-business but also because it allows us to showcase different features of the new SweetRules V2 implementation. Specifically, we show how powerful features of the new implementation such as importing OWL-DLP ontologies into a rule-based knowledge base, executing real-world business processes such as sending e-mail from rules, and inferencing on RuleML rules obtained from ontologies as well as rulebases possibly expressed in different types of KR. The procurement example allows us to also see how different business functions/features such as rebates, financing scenarios, payment options, which might be applicable in a wide variety of business applications, can be expressed using the RuleML KR language.

From our investigation of the electronic procurement scenario, we suggest inclusion of specific features in future versions of the RuleML KR to support query and message management that would be useful especially in business applications involving iterated request-response communication, such as e-contracting applications. Finally, we also explain how our electronic contracting approach based on RuleML and SweetRules can relate to other e-business technologies such as RosettaNet [4] and ebXML [5].

The paper is organized as follows. In section 2, we provide a brief overview of the technologies – RuleML, SweetRules, and SweetDeal – that we use in this research. Section 3 provides an overview of our approach and scenario. Section 4 illustrates the expressive power of RuleML in representing key contract provisions, specifically those of financial incentives. Section 5 describes the iterated contract construction process in great detail. Section 6 concludes the paper.

2 Overview of Technologies

We provide below a short description of the different technologies used in this research.

2.1 RuleML

RuleML [1] is the emerging standard for representing semantic web rules. The fundamental KR used in RuleML is situated courteous logic program or SCLP, which has been demonstrated to be expressively powerful [6]. The courteous part of SCLP enables prioritized conflict handling, which in turn enables modularity in specification, modification, merging and updating. The situated part of SCLP enables attached procedures for "sensing" (i.e. testing rule antecedents) and "effecting" (i.e. performing actions when certain conclusions are reached).

2.2 SweetRules

SweetRules [2], a uniquely powerful integrated set of tools for semantic web rules and ontologies, is newly enhanced in V2.1. The new version of SweetRules include capabilities such as first-of-a-kind semantics-preserving translation and interoperability between a variety of rule and ontology languages (including XSB Prolog [7], Jess [8] production rules, HP Jena-2 [9], IBM CommonRules [10], and the SWRL [11] subset of RuleML), highly scaleable backward and forward inferencing, and easy merging of heterogeneous distributed rulebases/ontologies.

2.3 SweetDeal

SweetDeal [12] is an electronic contracting approach that uses SCLP RuleML to support creation, evaluation, negotiation, execution and monitoring of formal electronic contracts between agents such as buyers and sellers. The approach builds on top of the SweetRules toolest to showcase the power of SCLP, RuleML, and SweetRules, as a design -- and implemented prototype software -- in the specific business application of electronic contracting.

3 Overview of Approach and Scenario

The extended SweetDeal approach described in this paper consists of three primary pieces: communication protocol between the contracting agents, contract knowledge bases and agent communication knowledge bases. We briefly describe these below in the context of our specific scenario of electronic procurement.

3.1 Communication Protocol

In our scenario, the buyer, Acme Corp, is interested in purchasing computers of a particular configuration. The buyer attempts to establish a procurement contract with the seller, Dell Computers. We assume that Dell Computers is a preferred vendor of computers for Acme Corp. To establish the terms of the contract, the buyer and seller agents exchange messages in an iterated fashion.

The protocol of message exchanges is as follows: the buyer first sends an RFP (request for proposal) to the seller. The seller responds to the RFP with the proposal. Based on specific business criteria, the buyer chooses to accept or reject the proposal. The buyer may also suggest modifications to the proposal before

accepting or rejecting it. The RFP message from the buyer contains specific details about the desired computer configuration. It also contains any queries to which the seller must provide answers in its proposal. The proposal message from the seller contains several formal contract fragments which describe useful business provisions such as rebates, financing options, as well as payment options for the buyer. In addition to specifying the contractual provisions, the seller also provides answers to the queries posed by the buyer. Finally, it may pose additional queries for the buyer that the buyer in turn must provide answers to in the next negotiation message. After the buyer is satisfied with the final contract proposal from the seller, it generates a purchase order that is sent to the seller. To complete the transaction, the seller delivers the order and the buyer makes arrangements to pay the seller via the chosen payment option. Any contingencies in the execution of the order/transaction are handled according to the terms of the contract.

3.2 Contract Knowledge Bases

Contract negotiation messages exchanged between the agents are RuleML knowledge bases that are executable within SweetRules V2.1 software. Contract knowledge bases contain the following six main technical components: rules, facts, ontologies including OWL-based ontologies as well as object-oriented default inheritance ontologies, effectors, f-queries and their answers, and conditional queries. We briefly describe each of these components below. Since RuleML as an XML-based markup language is fairly verbose and since the presentation syntax of RuleML has not yet been implemented completely in SweetRules, we use the IBM CommonRules (CR) V3.3 syntax in all our examples to allow for concise presentation and easier comprehension. In future, it would be more desirable instead to use the RuleML presentation syntax. See [16], especially the Rules language description, for the initial version of that presentation syntax, and see [2], especially its documentation, for its experimental extension to include the Situated feature and for its (currently, still partial) support in SweetRules.

3.2.1 Rules

RuleML rules express the if-then implications of the contractual fragments and form the bulk of the contract knowledge base. Each rule has a head and a body. The “head” is the part of the rule after the “then”, whereas the “body” is the part of the rule that follows “if” and precedes “then”. The example below shows a simple <rebate> rule: the seller might wish to provide a rebate offer to the buyer in the proposal. Specifically, the seller might wish to offer a rebate in the amount of $1000 to the buyer if the number of computers ordered by the buyer is more than 75. Due to current tool limitations of numeric types in translating CommonRules to RuleML, all numeric constants in the rule examples below are represented using strings, e.g., “75” is represented as “seventyfive”.

```
<rebate>
if
       quoteID(?QuoteID) AND quantityOfItemOrdered(?Q) AND
              isGreaterThan(?Q, seventyfive)
then
       rebateAmount(?QuoteID, thousand);
```

3.2.2 Facts

RuleML facts or assertions are rules that have no bodies. The simple examples below show facts that are specified in the RFP from the buyer to the seller. The quantity of item ordered by the buyer is 80 (computers) and the buyer is located in the state of Florida. (We assume that both buyer and seller are located in USA).

```
       quantityOfItemOrdered(eighty);
       buyerLocationState(florida);
```

3.2.3 Ontologies

Ontologies are vocabularies that express the background knowledge used by the contract rules. They can be either OWL [15] ontologies or rule-based object-oriented default inheritance ontologies. OWL ontologies used must be in the Description Logic Programs (DLP) [13] subset of OWL, i.e. in the subset of OWL that is translatable into LP rules. SweetRules V2.1 software allows for translation from OWL-DLP to RuleML rules. We show below a simple example of an OWL ontology that is used by the buyer. The ontology (procurement.owl) has three classes: *buyer*, *seller*, and *product*, and three object properties: *preferredVendorIs*, *buysProduct*, and *sellsProduct*. The ontology fragment also has some instance data: *computers is a product*, *Dell sells computers*, *Acme buys computers*, *Acme has Dell as a preferred vendor*. Since the ontology is in the DLP subset of OWL, a translation from OWL to RuleML exists and SweetRules V2.1 software can be used (see command C1 below) to convert the ontology to a rule-based knowledge base in RuleML.

```
translate owl clp c:\procurement.owl c:\procurement.clp          (C1)
```

The ontology (procurement.owl) is shown below:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.procurement.org/procurement.owl#"
    xml:base="http://www.procurement.org/procurement.owl">
  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="buyer"/>
  <owl:Class rdf:ID="seller"/>
  <owl:Class rdf:ID="product"/>

  <owl:ObjectProperty rdf:ID="preferredVendorIs">
        <rdfs:domain rdf:resource="#buyer"/>
        <rdfs:range rdf:resource="#seller"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="buysProduct">
        <rdfs:domain rdf:resource="#buyer"/>
        <rdfs:range rdf:resource="#product"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="sellsProduct">
        <rdfs:domain rdf:resource="#seller"/>
        <rdfs:range rdf:resource="#product"/>
  </owl:ObjectProperty>

  <seller rdf:ID="dell">
       <sellsProduct rdf:resource="#computers"/>
  </seller>

  <buyer rdf:ID="acme">
        <preferredVendorIs rdf:resource="#dell"/>
         <buysProduct rdf:resource="#computers"/>
  </buyer>

  <product rdf:ID="computers"/>
</rdf:RDF>
```

The translation of the ontology to rules is shown below. The translation has been slightly modified for ease of readability. Each of the predicates below would be prefixed in the original translation with a long

namespace URI indicated in the OWL document above. The namespace URI has been removed from all predicates below.

```
<emptyLabel> if buysProduct(?X, ?Y) then buyer(?X);
<emptyLabel> if buysProduct(?X, ?Y) then product(?Y);
<emptyLabel> if sellsProduct(?X, ?Y) then seller(?X);
<emptyLabel> if sellsProduct(?X, ?Y) then product(?Y);
<emptyLabel> if preferredVendorIs(?X,?Y) then buyer(?X);
<emptyLabel> if preferredVendorIs(?X, ?Y) then seller(?Y);
<emptyLabel> sellsProduct(dell, computers);
<emptyLabel> preferredVendorIs(acme, dell);
<emptyLabel> buyer(acme);
<emptyLabel> product(computers);
<emptyLabel> Class(product);
<emptyLabel> Class(buyer);
<emptyLabel> Class(seller);
<emptyLabel> seller(dell);
<emptyLabel> buysProduct(acme, computers);
```

Next we show a simple example of expressing an object-oriented default inheritance ontology using rules. In the example, *BuyWithCredit* is a subclass of *Buy*. *Buy* assigns the value "invoice" to the *paymentMode* property, but *BuyWithCredit* assigns the value "credit" to the *paymentMode* property, i.e., *BuyWithCredit* overrides the *paymentMode* property inherited by default from *Buy*. The courteous feature of SCLP RuleML is a powerful way to express default inheritance using rules. If only Buy(quoteID) is asserted (i.e. the buyer asserts that it wants to buy), then the payment mode is assumed to be invoice (by default). If the buyer specifically asserts BuyWithCredit(quoteID), then the default payment mode is overridden to be credit instead.

```
<buyRegular>if Buy(?quoteID) then paymentMode(?quoteID,invoice);
/* BuyWithCredit is a subclass of Buy */
if BuyWithCredit(?quoteID) then Buy(?quoteID);
<buyCredit> if BuyWithCredit(?quoteID)then paymentMode(?quoteID,credit);
overrides(buyCredit, buyRegular);
```

3.2.4 Effectors

Effectors are a feature of the Situated extension of logic programs. An effector procedure is an attached procedure that is associated with a particular predicate. This association is specified via an effector statement that is part of the rulebase. When a conclusion is drawn about the predicate, an action is triggered; this action is the invocation of the effector procedure, and is side-effect-ful. In general, there may be multiple such effector statements and procedures in a given rulebase, e.g.., in a given SweetDeal contract/proposal. Effectors can execute real-world business processes associated with the execution of the contract. For example, an effector can be used by the buyer to send the purchase order (PO) to the seller (see <sendPO> rule below). If the vendor proposal has been approved, then the buyer sends the PO to the sales e-mail address of the vendor. The effector *sendPOtoVendor* is associated with the Java procedure *emailMessage* in the *Effector_EmailPO* class, whose path is indicated as *com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs*.
The Java procedure not shown here for brevity handles the e-mail messaging aspect of sending the PO to the vendor. The arguments to the effector predicate – seller e-mail address, location of the purchase order, approved proposal identifier – are passed as arguments to the Java procedure.

```
<sendPO>
if
  approvedVendorProposal(?Vendor, ?ProposalID) AND
  emailSalesAddress(?Vendor, ?SellerAddress) AND locationOfPO(?Location)
then
  sendPOtoVendor(?SellerAddress, ?Location, ?ProposalID);
```

```
<emptylabel>
   Effector: sendPOtoVendor
   Class: Effector_EmailPO
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

3.2.5 Fact-queries or F-queries

The traditional notion of the answer to a query in logic programs (and databases) is: a set of variable-binding lists. In modeling the exchange of contract proposals and associated dialogue between contracting parties, however, it is often convenient to model the answer to an inquiry as a set of facts instead. Accordingly, we have developed the design of f-queries (short for "fact queries") as a (fairly simple) experimental extension to RuleML. Note that, unlike the rest of what we describe of the SweetDeal approach in this paper, this f-queries feature is *not yet implemented in SweetRules*. RuleML f-queries are queries which have facts as their answers. They facilitate the iterated development of procurement contracts. The example below shows a sample f-query. It is an f-query from buyer to seller in which the buyer requests the seller for the unitPriceOfItem. The answer to the f-query is provided by the seller as a RuleML fact.

```
Query Example
<query>
  <_body>
    <fclit cneg="no" fneg="no">
      <_opr>
        <rel>unitPriceOfItem</rel>
      </_opr>
      <var>QuoteID</var>
      <var>Price</var>
    </fclit>
  </_body>
</query>
```

3.3 Agent Communication Knowledge Bases

In addition to the contract knowledge bases that are shared/exchanged, the agents also have internal RuleML knowledge bases that contain rules to facilitate agent communication. The effectors feature of SCLP RuleML allows the agents to execute real-world business processes such as e-mail messaging. This feature is used by the agents to send the contract rulesets to each other. The actual e-mail messaging effector procedure is implemented as a Java method that employs the JavaMail API [14]. The communication process is triggered using the internal agent communication KB and the SweetRules V2.1 software that supports execution of Java methods attached as effectors to specified predicates in the KB. A simple example follows: the situated rule <sendRFP> allows the buyer to send the RFP ruleset to the sales e-mail address of the seller. The name of the effector in the situated rule is sendRFPtoComputerSeller. The effector specification consists of the name of the Java procedure (emailMessage), the Java implementation class that contains the method (Effector_EmailRFP), and the path to the class (com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs).

The effector is executed when the buyer wants to buy computers and the seller sells computers and is in the preferred vendor list of the buyer. When the sendRFPtoComputerSeller predicate is concluded, the attached procedure "emailMessage" is called to execute the required action. The action consists of reading the RFP from the local file system and sending it via e-mail to the specified e-mail address of the sales department of the seller. For brevity, the Java code to implement the e-mail messaging is not shown here.

```
<sendRFP>
if
   wantToBuy(?Buyer, computers) AND seller(?Vendor) AND
```

```
       sell(?Vendor, computers) AND inPreferredVendorList(?Buyer, ?Vendor) AND
       emailSalesAddress(?Vendor, ?Address) AND
       locationofRFP(?Buyer, computers, ?Location)
then
   sendRFPtoComputerSeller(?Address, ?Location);

<emptylabel>
   Effector: sendRFPtoComputerSeller
   Class: Effector_EmailRFP
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

4 Contract Business Provisions using RuleML

In this section, we present a few key contract fragments in the procurement contracting scenario and how SCLP RuleML can be used to express them. We intend to show how the expressive/declarative power of RuleML allows for easy addition and modification of key B2B contracting provisions. Specifically, we focus on expressing commonly used financial incentives such as rebates, discount pricing, and financing options. These incentives could be specified by the seller in its proposal. For the sake of simplicity and brevity, in this paper version some of the rules (e.g., about monthly payments in financing options) are highly specific to the particular scenario, rather than specified in more realistically general form.

4.1 Rebate

For example: the seller wishes to offer a rebate in the amount of $1000 to the buyer if the quantity of item ordered is greater than 75. This is represented as the <rebate> rule below.

```
<rebate>
if
       quoteID(?QuoteID) AND quantityOfItemOrdered(?Q) AND
       isGreaterThan(?Q, seventyfive)
then
       rebateAmount(?QuoteID, thousand);
```

4.2 Pricing Options

For example: If the buyer makes the purchase before April 1 then the unit price offered by the seller is $600; if the purchase is made before April 15, then the unit price offered is $650. This is specified as the <earlyPurchase> and <latePurchase> rules below. If both these rules apply, i.e., if the purchase was made before April 1, then precedence is given to the earlyPurchase rule. This precedence is specified using the courteous prioritization feature of SCLP (and of RuleML): see the overrides fact rule below.

```
<earlyPurchase>
if
       quoteID(?QuoteID) AND purchaseDate(?QuoteID, ?Date) AND
       isLessThan(?Date, oneApr05)
then
       unitPriceOfItem(?QuoteID, sixhundred);

<latePurchase>
if
       quoteID(?QuoteID) AND purchaseDate(?QuoteID, ?Date) AND
       isLessThan(?Date, fifteenApr05)
then
       unitPriceOfItem(?QuoteID, sixhundredfifty);

overrides(earlyPurchase, latePurchase);
```

```
MUTEX
       unitPriceOfItem(?QuoteID, sixhundred) AND
       unitPriceOfItem(?QuoteID, sixhundredfifty);
```

4.3 Financing Option

For example: If the financing is requested for 36 months by the buyer, the unit price of the item is determined to be $600, and the quantity ordered is 50, then the financing option offered by the seller is such that the monthly payment is $958 and the total interest paid is $4500 (see the <financing> rule below).

```
<financing>
if
       quoteID(?QuoteID) AND financeForMonths(?QuoteID, thirtysixMonths) AND
       unitPriceOfItem(?QuoteID, sixhundred) AND
       quantityOfItemOrdered(?QuoteID, fifty)
then
       monthlyPayment(?QuoteID, ninehundredfiftyeight) AND
       totalInterest(?QuoteID, fourthousandfivehundred);
```

5 Details of Procurement Contract Construction Using RuleML and SweetRules V2.1

In this section, we describe in detail the specific steps taken in constructing an e-contract between the buyer and seller using SCLP RuleML and SweetRules V2.1 in our electronic procurement scenario.

As described earlier, the buyer has a solo (or unshared) agent communication knowledge base that can be used to initiate the action of sending an RFP to a specific seller (in our example – Dell). We call this solo knowledge base – BSO1. BSO1 has the names of the different sellers, types of products offered by them, their respective sales e-mail addresses, and whether the sellers are in the preferred vendor list maintained by the buyer. The location of the RFP (which itself is a rule-based knowledge base) is indicated using the *locationofRFP* predicate. The rule that triggers sending the RFP to the seller is indicated by <sendRFP>: if the buyer wants to buy computers and the seller sells computers and is in the preferred vendor list of the buyer, send the RFP from the indicated local filesystem location to the seller's sales e-mail address. The predicate *sendRFPtoComputerSeller* is associated with the situated effector procedure *emailMessage*, which uses the JavaMail API to send the RFP ruleset to the seller via e-mail.

Buyer Solo KB – BSO1

```
wantToBuy(acme, computers);
seller(dell);
seller(staples);
sell(dell, computers);
sell(staples, officesupplies);
inPreferredVendorList(acme, dell);
inPreferredVendorList(acme, staples);
emailSalesAddress(dell, "sales@dell.com");
emailSalesAddress(staples, "sales@staples.com");
locationofRFP(acme, computers, "c:\\buyertosellerRFP.clp");

<sendRFP>
if
   wantToBuy(?Buyer, computers) AND seller(?Vendor) AND
  sell(?Vendor, computers) AND inPreferredVendorList(?Buyer, ?Vendor) AND
  emailSalesAddress(?Vendor, ?Address) AND
  locationofRFP(?Buyer, computers, ?Location)
then
  sendRFPtoComputerSeller(?Address, ?Location);
```

```
<emptylabel>
   Effector: sendRFPtoComputerSeller
   Class: Effector_EmailRFP
   Method: emailMessage
   path:
"com.ibm.commonrules.examples.situated_programming_examples.familymsg.aprocs";
```

In SweetRules V2.1, the "exhaustForwardInfer" command is given to derive all the conclusions from a given rulebase, and along with those conclusions to perform all the associated effecting actions that those conclusions trigger (i.e., sanction). For example, the command C2 below generates all the conclusions of BSO1 and (as an effecting action) sends the RFP to the seller. The "clp" in the first two arguments of the command indicates that CommonRules V3.3. format is the input and output knowledge base format, the third argument gives the location of BSO1, and the fourth argument specifies that IBM CommonRules should be used indirectly as an underlying inference engine when performing inferencing. SweetRules V2.1 software allows for a choice of such underlying engines. In our example, SweetRules enables Jess or XSB, as well as CommonRules, to be used as indirect underlying engine; for each choice of underlying engine, it would generate semantically equivalent conclusions and perform the same set of triggered effecting actions

```
exhaustForwardInfer clp clp c:\buyertosellerSendRFP.clp CommonRules      (C2)
```

The RFP sent by the buyer to the seller is a collection of rules. The RFP consists of two parts -- a shared knowledge base that contains most importantly the required computer configuration details (we call this knowledge base BSH1) and a set of f-queries that request specific answers from the seller (we call this set of queries BFQ1).

BSH1 indicates the buyer name, quantity of item ordered, buyer state, and the required computer configuration details. The rule <checkOfferedConfiguration> is used by the buyer to check whether the vendor offered configuration satisfies the minimum requirements. Since RuleML built-ins are not currently directly and smoothly supported in SweetRules V2.1 beyond the SWRL subset of RuleML, we also provide several facts to support arithmetic comparison.

Buyer to Seller RFP (BSH1)

```
buyerName(acme);  /* buyer name is acme */
quantityOfItemOrdered(fifty);  /* quantity of item ordered is fifty */

/* buyer is located in the state of Florida */
buyerLocationState(florida);

/* speed of processor should be at least 2GHz */
requiredMinProcessorSpeedInGHZ(twogigahertz);
if
 requiredMinProcessorSpeedInGHZ(?Speed) and
 offeredProcessorSpeedInGHZ(?OfferSpeed) and isGreaterThan(?OfferSpeed, ?Speed)
then
 isSpeedAcceptable(true);

/* not shown here for brevity:  there are also additional computer system
configuration details (memory size, hard disk storage capacity, monitor size,
monitor type (flat?), monitor resolution) */ ...
...

/* check if the configuration is acceptable */
<checkOfferedConfiguration>
if
  isSpeedAcceptable(true) and isMemorySizeAcceptable(true) and
  isHardDiskSizeAcceptable(true) and isMonitorSizeAcceptable(true) and
  offeredMonitorType(flat) and
  offeredMonitorResolution(tenTwentyFourBySevenSixtyEight)
then
  isOfferedConfigurationAcceptable(true);
```

```
/* The following are some facts in lieu of arithmetic built-ins. */
isGreaterThan(fourgigahertz, twogigahertz);
isGreaterThan(onezerotwofourmb, fivetwelvemb);
isGreaterThan(sixtyGB, fortyGB);
isGreaterThan(seventeen, fifteen);
```

BFQ1 is the collection of f-queries that ask the seller to specify the vendor quote identifier, the offered computer configuration details, the unit price of item, taxes as percent of price, service charge as percent of price, delivery charges for shipment, and the delivery time in days. For brevity, only a few of the f-queries are shown below.

Buyer to Seller f-Queries (BFQ1)

```
<rulebase>
  <_rbaselab>
    <ind>FQueries</ind>
  </_rbaselab>
  <query>
    <_body>
      <fclit cneg="no" fneg="no">
        <_opr>
          <rel>quoteID</rel>
        </_opr>
       <var>QuoteID</var>
      </fclit>
    </_body>
  </query>
  <query>
    <_body>
      <fclit cneg="no" fneg="no">
        <_opr>
          <rel>offeredProcessorSpeedInGHZ</rel>
        </_opr>
       <var>Speed</var>
      </fclit>
    </_body>
  </query>
...
```

After the seller receives the RFP, the seller sends its rule-based contract proposal to the buyer. The proposal contains three parts – BSH1 (i.e. shared knowledge base transmitted from buyer to seller – see above), answers to f-queries posed by the buyer plus the shared knowledge base that contains rules about pricing, rebates, financing options and other business provisions (we call this SSH1), and lastly f-queries for the buyer (SFQ1).

Seller to Buyer (SSH1)

```
/* quote ID is 1 */
quoteID(one);
/* computer configuration details */
offeredProcessorSpeedInGHZ(fourgigahertz);
offeredSizeofmemoryInMB(onezerotwofourmb);
offeredSizeofharddiskInGB(sixtyGB);
offeredMonitorSizeInInches(seventeen);
offeredMonitorType(flat);
offeredMonitorResolution(tenTwentyFourBySevenSixtyEight);

/* Pricing Rules */
/* if purchase date is before April 1 2005, then unit Price is $600;
   if purchase date is before April 15 2005, then unit Price is $650*/
<earlyPurchase>
if
```

```
       quoteID(?QuoteID) and purchaseDate(?QuoteID, ?Date) and
       isLessThan(?Date, oneApr05)
then
       unitPriceOfItem(?QuoteID, sixhundred);
<latePurchase>
if
       quoteID(?QuoteID) and purchaseDate(?QuoteID, ?Date) and
       isLessThan(?Date, fifteenApr05)
then
       unitPriceOfItem(?QuoteID, sixhundredfifty);

overrides(earlyPurchase, latePurchase);

MUTEX
       unitPriceOfItem(?QuoteID, sixhundred) and
       unitPriceOfItem(?QuoteID, sixhundredfifty);

/* there is no service charge */
if
       quoteID(?QuoteID)
then
       serviceChargeAsPercentOfPrice(?QuoteID, zeroPercent);

/* Delivery Options */

/* if delivery type is standard then delivery charge is $2500 for the order
   if delivery type is express then delivery charge is $5000 for the order
*/
<standard>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, standard)
then
       deliveryChargesForShipment(?QuoteID, twentyfivehundred);

<express>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, express)
then
       deliveryChargesForShipment(?QuoteID, fivethousand);
MUTEX
       deliveryType(?QuoteID, standard) and deliveryType(?QuoteID, express);

/* if delivery type is standard then delivery time in days is 14 days
   if delivery type is express then delivery time in days is 7 days
*/
<standardDeliveryTime>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, standard)
then
       deliveryTimeInDays(?QuoteID, fourteendays);

<expressDeliveryTime>
if
       quoteID(?QuoteID) and deliveryType(?QuoteID, express)
then
       deliveryTimeInDays(?QuoteID, sevendays);

MUTEX
       deliveryTimeInDays(?QuoteID, fourteendays) and
deliveryTimeInDays(?QuoteID, sevendays);

/* Additional assertions from Seller */
/* Financial Incentives section */
```

```
/* not shown here for brevity: the financial incentives of discount pricing,
rebate, financing option already shown above in section 4 */
...
/* Sales Tax */
/* no sales tax in Florida */
<tax0>
if
       quoteID(?QuoteID) and buyerLocationState(florida)
then
       taxesAsPercent(?QuoteID, zeroPercent);

/* 5% sales tax in states other than Florida */
<tax5>
if
       quoteID(?QuoteID) and buyerLocationState(?X) and NotEquals(?X, florida)
then
       taxesAsPercent(?QuoteID, fivePercent);

MUTEX
       taxesAsPercent(?QuoteID, zeroPercent) and
       taxesAsPercent(?QuoteID, fivePercent);

/* Object-oriented default inheritance using rules */
/* If you buy, then default payment mode is invoice */
<buyRegular> if Buy(?QuoteID) then
paymentMode(?QuoteID, invoice);

/* BuyWithCredit is a subclass of Buy */
if BuyWithCredit(?QuoteID) then Buy(?QuoteID);

<buyCredit>
if BuyWithCredit(?QuoteID) then paymentMode(?QuoteID, credit);

overrides(buyCredit, buyRegular);

MUTEX
       paymentMode(?QuoteID, credit) and paymentMode(?QuoteID, invoice);

isLessThan(twentyfiveMarch05, oneApr05);
isLessThan(twentyfiveMarch05, fifteenApr05);
isLessThan(fiveApr05, fifteenApr05);
isGreaterThan(eighty, seventyfive);
NotEquals(massachusetts, florida);
```

SFQ1 is a collection of f-queries posed by the seller for the buyer. The seller asks whether the buyer would like to buy and whether the buyer would like to buy with a credit card. The seller also queries for the purchase date, delivery type and number of months of financing requested. For brevity, only a few of the f-queries are shown below.

Seller to Buyer F-Queries (SFQ1)

```
<rulebase>
  <_rbaselab>
  <ind>FQueries</ind>
  </_rbaselab>
  <query>
    <_body>
       <fclit cneg="no" fneg="no">
         <_opr>
              <rel>purchaseDate</rel>
          </_opr>
         <var>QuoteID</var>
         <var>Date</var>
```

```
        </fclit>
    </_body>
  </query>
...
```

When the buyer receives the proposal ruleset from the seller, it answers the queries posed by the seller (see BA1 below) and then performs exhaustive inferencing on the resulting ruleset (BSH1 + SSH1 + BA1) to obtain the derived conclusion set (CS1). Logical inferencing allows the buyer to determine the key parameters (such as unit price, delivery charges, taxes, etc.) of the proposal and also whether the proposal meets minimum specified criteria in the RFP.

Answers to F-Queries posed by seller (BA1)

```
Buy(one);
BuyWithCredit(one);
purchaseDate(one, fiveApr05);
deliveryType(one, express);
financeForMonths(one, thirtysixMonths);
```

The conclusion set (CS1) tells the buyer that the offered configuration is acceptable, unit price of item will be $650, delivery time will be 7 days, % discount already included in the price is 13%, taxes are 5%, rebate amount is $1000, and payment mode is credit.

Conclusion Set (CS1) obtained from BSH1 + SSH1 + BA1

```
isLessThan(twentyfiveMarch05, oneApr05);
isLessThan(twentyfiveMarch05, fifteenApr05);
isLessThan(fiveApr05, fifteenApr05);
requiredMinProcessorSpeedInGHZ(twogigahertz);
quoteID(one);
requiredMinSizeofmemoryInMB(fivetwelvemb);
offeredSizeofmemoryInMB(onezerotwofourmb);
requiredMonitorResoluton(tenTwentyFourBySevenSixtyEight);
purchaseDate(one, fiveApr05);
quantityOfItemOrdered(eighty);
BuyWithCredit(one);
deliveryType(one, express);
NotEquals(massachusetts, florida);
isGreaterThan(fourgigahertz, twogigahertz);
isGreaterThan(onezerotwofourmb, fivetwelvemb);
isGreaterThan(sixtyGB, fortyGB);
isGreaterThan(seventeen, fifteen);
isGreaterThan(eighty, seventyfive);
creditCardNumber(one, ccNumber9876543298765432);
offeredSizeofharddiskInGB(sixtyGB);
overrides(earlyPurchase, latePurchase);
overrides(earlyPurchaseDiscount, latePurchaseDiscount);
overrides(buyCredit, buyRegular);
offeredMonitorSizeInInches(seventeen);
requiredMinSizeofharddiskInGB(fortyGB);
offeredProcessorSpeedInGHZ(fourgigahertz);
financeForMonths(one, thirtysixMonths);
requiredMonitorType(flat);
offeredMonitorType(flat);
buyerName(acme);
buyerLocationState(massachusetts);
requiredMinMonitorSizeInInches(fifteen);
offeredMonitorResolution(tenTwentyFourBySevenSixtyEight);
vendorName(dell);
serviceChargeAsPercentOfPrice(one, zeroPercent);
deliveryChargesForShipment(one, fivethousand);
isSpeedAcceptable(true);
Buy(one);
```

```
rebateAmount(one, thousand);
isMonitorSizeAcceptable(true);
isMemorySizeAcceptable(true);
isHardDiskSizeAcceptable(true);
isOfferedConfigurationAcceptable(true);
deliveryTimeInDays(one, sevendays);
discountPercentAlreadyIncluded(one, thirteen);
unitPriceOfItem(one, sixhundredfifty);
taxesAsPercent(one, fivePercent);
paymentMode(one, credit);
```

6 Relationship of other B2B Technologies to our Approach

RosettaNet and ebXML are two very important and influential approaches to XML-based e-business messaging including about contracting and e-commerce. It is desirable to be able to use our SweetDeal approach together with such XML-based e-business messaging infrastructure. In this section, we discuss how SweetDeal and (SCLP) RuleML can be used with RosettaNet and with ebXML. The punchline is that they play well together; the SweetDeal contract rulesets can be carried as the "letters" content within the "envelopes" of RosettaNet or ebXML messages, i.e., within their messaging interfaces and protocols. In doing so, it is both possible and useful to utilize the (non-OWL) ontologies provided by RosettaNet and ebXML, and to perform sending of messages as actions.

6.1 RosettaNet

Next, we begin with RosettaNet, and discuss specifically how RosettaNet Partner Interface Processes (PIPs) can be used with RuleML in the context of our electronic procurement scenario. RosettaNet is a consortium of information technology, electronic components, semiconductor manufacturing and solutions providers, which seeks to establish a common language and standard processes for business-to-business (B2B) transactions. RosettaNet PIPs define business processes between trading partners. The PIP specifies the roles of the trading partners that participate in the business process as well as the business activities that compose the process. The PIP also specifies XML-based action messages or business documents that are exchanged between the roles during business activities. The specification of a standard structure for the business documents is a major part of the PIP specification. An example of a RosettaNet PIP is PIP3A1 which provides a detailed XML message guideline for implementing the Request Quote business process. A message fragment from PIP3A1 is shown below –

```
<ContactInformation>
     <contactName>
            <FreeFormText>A</FreeFormText>
            <EmailAddress>abc@xyz.com</EmailAddress>
            …..
     </contactName>
</ContactInformation>
```

The message fragment above specifies the structure for contact information for the buyer who sends the request for quote to the seller. Our SweetDeal approach can be used straightforwardly in combination with the exchange of RosettaNet PIP messages between the two parties. We can also directly use the standardized (non-OWL) ontological terms from the PIP messages in our rulebases. For example, the request for proposal (RFP) sent by the buyer to the seller in our scenario allows for use of the ontological terms in the RosettaNet PIP3A1 XML message guidelines. A SweetDeal quote (contract proposal) rulebase cf. our earlier scenario can then employ as predicates (i.e., as ontological terms) various properties drawn from the PIP specification, e.g., the unit price of the product, which is specified in RosettaNet using the following DTD segment –

```
<!ELEMENT unitPrice ( ProductPricing ) >
<!ELEMENT ProductPricing ( FinancialAmount , GlobalPriceTypeCode ) >
<!ELEMENT FinancialAmount ( GlobalCurrencyCode , MonetaryAmount ) >
<!ELEMENT GlobalCurrencyCode ( #PCDATA ) >
<!ELEMENT MonetaryAmount ( #PCDATA ) >
```

For example, the seller would specify the following fact rule in the proposal to the buyer:

```
unitPrice(?GlobalCurrencyCode, ?MonetaryAmount).
```

6.2 ebXML

Likewise, ebXML can be used in our scenario along with RuleML and the SweetDeal approach to support electronic contracting between two parties. Both the buyer and the seller in our scenario would maintain ebXML collaboration protocol profiles (CPPs) that would describe the specific business collaborations supported by each of the parties using the ebXML business process specification schema (BPSS). For example, the buyer CPP would show that the "request for proposal" is a business process that is supported by it. The details of the "request for proposal" business process would be specified using the ebXML BPSS. The parties that will engage in the interaction protocol will reach agreement on how to collaborate by exchanging the CPPs to construct a collaboration protocol agreement (CPA), which fixes the protocol for interaction between the parties. Once agreement has been reached, ebXML messages in accordance with the collaboration agreement can be exchanged using ebMS (or ebXML Message Service). The payload of these messages can contain the RuleML rulebases to establish the electronic procurement contract.

7 Conclusions

In this paper, we have extended the SweetDeal approach and applied the extended approach using the new SweetRules V2.1 semantic web rules prototype software to a practical, real-world B2B application in the domain of electronic contracting. The electronic procurement contracting scenario that we have described in detail shows how semantic web rules technology, specifically RuleML and SweetRules, can be powerfully used in e-contracting.

Acknowledgements: Thanks to Shashidhara Ganjugunte, Chitravanu Neogy, Said Tabet, and the rest of the SweetRules development team, for helping to realize the implementation, and for useful discussions.

References

[1] Rule Markup Language Initiative, http://www.ruleml.org and http://www.mit.edu/~bgrosof/#RuleML
[2] SweetRules, http://sweetrules.projects.semwebcentral.org/
[3] OWL and the Semantic Web Activity of the World Wide Web Consortium. http://www.w3.org/2001/sw
[4] RosettaNet, http://www.rosettanet.org
[5] ebXML (ebusiness XML) standards effort, http://www.ebxml.org
[6] Grosof, B.N., "Representing E-Business Rules for Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML". Proc. Wksh. on Information Technology and Systems (WITS '01), 2001.
[7] XSB logic programming system. http://xsb.sourceforge.net/
[8] Jess (Java Expert System Shell). http://herzberg.ca.sandia.gov/jess/
[9] Jena, http://jena.sourceforge.net/
[10] IBM CommonRules. http://www.alphaworks.ibm.com and http://www.research.ibm.com/rules/
[11] SWRL, A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
[12] Grosof, B.N., C.T. Poon, "SweetDeal: Representing Agent Contracts With Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions". International Journal of Electronic Commerce (IJEC), 8(4):61-98, Summer 2004, Special Issue on Web E-Commerce.
[13] Grosof, B.N., Horrocks, I., Volz, R., and Decker, S., "Description Logic Programs: Combining Logic Programs with Description Logic". Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003).
[14] JavaMail, http://java.sun.com/products/javamail/
[15] OWL Web Ontology Language, http://www.w3.org/TR/owl-features/
[16] Semantic Web Services Framework Version 1.0, http://www.daml.org/services/swsf/1.0

The Taga agent framework

Travel Agent Game in Agentcities (TAGA) is the framework that extends and enhances the Trading Agent Competition (TAC) scenario to work in Agentcities, an open multi agent environment based on FIPA compliant platforms. TAGA uses the semantic web languages and tools (RDF and OWL) to specify and publish the underlying common ontologies; as a content language within the FIPA ACL messages; as the basis for agent knowledge bases via XSB-based reasoning tools; to describe and reason about services. TAGA extends the FIPA protocols to support open market auctions and enriches the Agentcities with auction services. The introducing of the semantic web languages improves the interoperability among agents. TAGA is intended as a platform for research in multi-agent systems, the semantic web and automated trading in dynamic markets as well as a self-contained application for teaching and experimentation with these technologies.

TAGA was used to demonstrate how semantic web technologies would fit into the popular FIPA multiagent systems framework. The reusable components include the FOWL reasoning system, and a number of ontologies designed support using OWL in a multi-agent systems environment.

Using Semantic web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment

TAGA is intended as a platform for research in multi-agent systems, the semantic web and automated trading in dynamic markets as well as a self-contained application for teaching and experimentation with these technologies. The following paper by Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan appeared in the Proceeding of the 5th International Conference on Electronic Commerce in September 2003.

F-OWL: an Inference Engine for the Semantic Web

F-OWL is an inference engine for the semantic web language OWL language based on F-logic, an approach to defining frame-based systems in logic. F-OWL is implemented using XSB and Flora-2 and takes full advantage of their features. We describe how F-OWL computes ontology entailment and compare it with other description logic based approaches. We also describe TAGA, a trading agent environment that we have used as a test bed for F-OWL and to explore how multiagent systems can use semantic web concepts and technology.

Using Semantic Web technology in Multi-Agent Systems: a case study in the TAGA trading agent environment

Youyong Zou, Tim Finin, Li Ding, Harry Chen, Rong Pan
U.Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD, 21250
410-455-3971

{yzou1,finin,dingli1,hchen4 }@cs.umbc.edu

ABSTRACT

Travel Agent Game in Agentcities (TAGA) is the framework that extends and enhances the Trading Agent Competition (TAC) scenario to work in Agentcities, an open multi agent environment based on FIPA compliant platforms. TAGA uses the semantic web languages and tools (RDF and OWL) to specify and publish the underlying common ontologies; as a content language within the FIPA ACL messages; as the basis for agent knowledge bases via XSB-based reasoning tools; to describe and reason about services. TAGA extends the FIPA protocols to support open market auctions and enriches the Agentcities with auction services. The introducing of the semantic web languages improves the interoperability among agents. TAGA is intended as a platform for research in multi-agent systems, the semantic web and automated trading in dynamic markets as well as a self-contained application for teaching and experimentation with these technologies.

Keywords

Agentcities, FIPA, Multi Agent System, OWL, Semantic Web, Trading Agent Competition.

1. INTRODUCTION

The Trading Agent Competition (TAC) [Wellman, 2002] was a test bed for intelligent software agents that interact through simultaneous auctions to obtain services for customers. The trading agents operated within the travel market scenario, buying and selling goods to best serve their given travel clients. TAC was designed to promote and encourage research in markets involving auction and autonomous trading agents and had proven to be successful after three consecutive year's competitions.

Although TAC's framework, infrastructure and game rules had evolved over the past three competitions [Stone, 2000] [Greenwald, 2001] [Wellman, 2002], the assumptions and approach of TAC limited its usefulness as a realistic test bed for agent based automated commerce. TAC used centralized market server as the sole mechanism for service discovery, communication, coordination, commitment, and control among the participating software agents. The trading agents communicate with the central auction server through network socket interface, exchanging pre-defined XML-based messages. In real world, the auction servers (for example, priceline.com and hotwire.com) and service providers are distributed among the massive open Internet and have distinct service descriptions and diverse service access interfaces. The abstractness and simplicity of the TAC approach helped to launch it as a research vehicle for studying bidding strategies, but are now perceived as a limiting factor for exploring the wide range of issues inherent in automated trading in open environment.

Agentcities [Dale, 2002] is the international initiative designed to explore the commercial and research potential of agent-based applications by constructing an open distributed network of platforms to host diverse agents and services. The ultimate goal is to enable the dynamic, intelligent and autonomous composition of services to achieve user and business tasks, therefore creating compound services to address changing needs. In such an open and distributed environment, the need of standard mechanisms and specifications is crucial for ensuring interoperability of distinct systems. The Foundation for Intelligent Physical gents (FIPA) produces such standards for heterogeneous and interacting agents and agent-based systems. In the production of these standards, FIPA promotes the technologies and interoperability specifications that facilitate the end-to-end inter-working of intelligent agent systems in modern commercial and industrial settings.

Inspired by TAC, we have developed Travel Agent Game in Agentcities (TAGA) on the foundation of FIPA technology and the Agentcities infrastructure. The agents and services use FIPA supported languages, protocols and service interfaces to create the travel market framework and provide stable communication environment where messages expressed in semantic languages can be exchanged. The travel market is the combination of auctions and varying markets including service registries, service brokerage, wholesalers, peer-to-peer transactions, bilateral negotiation, etc. This provides a richer test bed for experimenting with agents and web services as well as a interesting scenario to test and challenge agent technology. TAGA is running as a continuous open game at http://taga.umbc.edu/ and source code is available for research and teaching purposes.

The next section introduces the TAGA game and six types of agents. The details of using semantic web technology are presented in Section three. We discuss TAGA's features and our research

ICEC 2003. Pittsburgh, PA
ACM 1-58113-788-5/03/09

contributions in Section four and suggest the future works in Section five.

2. TAGA GAME AND AGENTS

We design TAGA as a general framework for running agent-based market simulations and games. Our first use of TAGA has been to build a travel competition along the lines that used in the last three year's TACs. In the competition, *customers* travel from City A to City B and spend several days before flying back. A *travel package* includes a round-trip flight ticket, corresponding hotel accommodation and tickets to entertainment events. A *travel agent* (an entrant to the game) competes with other travel agents in making contracts with customers and purchasing the limited travel services from the *Travel Service Agents*. Customer selects the travel agent with best travel itinerary. The objective of the travel agent is to acquire more customers, fulfill the customer's travel package, and maximize the profit.

TAGA provides a flexible framework to run the travel market game. Figure 1 show the structure of TAGA. The collaboration and competition among six types of agents who play different market roles simulate the real world travel market. We find that basing our implementation on FIPA compliant agent platforms has made the framework extremely flexible. We'll briefly describe the different agents in our initial TAGA game.

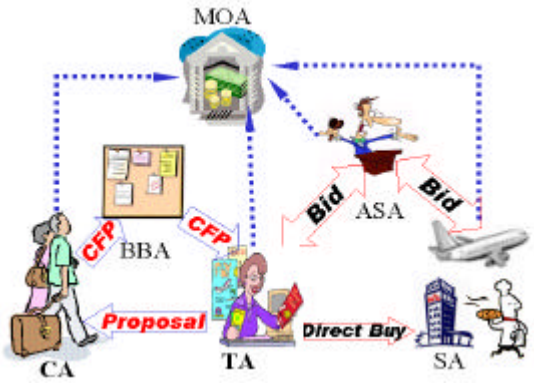

Figure 1: TAGA Architecture

The *Auction Service Agent* (ASA) operates all of the auctions in TAGA. Supported auction types include English and Dutch auctions as well as other dynamic markets similar to Priceline.com and Hotwire.com.

A *Service* Agent (SA) offers travel related service units such as airline tickets, lodging and entertainment tickets. Each class of travel related service has multiple providers with different service quality level and with limited service units. It allows other agents to query its description (e.g. service type, service quality, location) and its inventory (the availability or price of a certain type of service unit). Other agents may directly buy the service units through published service interface. SA also bids intentionally in the auctions to sell its good, e.g. listing its goods in auction and wait for the proper buyer.

A *Travel Agent* (TA) is a business that helps customers acquire travel service units and organize travel plan. The units can be bought either directly from the service agents, or through an auction server.

A *Bulletin Board Agent* (BBA) provides a mechanism helping customer agents find and engage one or more travel agents.

A *Customer Agent* (CA) represents an individual customer who has particular travel constraints and preferences. Its goal is to engage one or more TAs, negotiate with them over travel packages, and select one TA that is able to acquire all needed travel service units.

The *Market Oversight Agent* monitors the game and updates the financial model after each reported transaction and finally announces the winning TA when the game is over.

The basic cycle of the TAGA game has the following five stages:

- A customer-generating agent creates a new customer with particular travel constraints and preferences chosen from a certain distribution.
- The CA sends the customer's travel constraints and preferences to the BBA in the form of a CFP (call for proposal) message. The BBA forwards the CA's CFP message to each of the TAs that has registered with it. Each TA considers the CA's CFP independently and decides whether and how to respond.
- When deciding to propose a travel package, The TA contacts the necessary ASAs and SAs and assembles a travel itinerary. Note that the TA is free to implement a complex strategy using both aggregate markets (ASAs) as well as direct negotiation with SAs. The proposal to the CA includes the travel itinerary, a set of travel units, the total price and the penalty to be suffered by the TA if it is fail to complete the transaction.
- The CA negotiates with the TAs ultimately selecting one from which to purchase an itinerary based on its constraints, preferences and purchasing strategy (which might, for example, depend on a TA's reputation).
- Once the TA has a commitment from the CA, it attempts to purchase the units in the itinerary from the ASAs and SAs. There are two possible outcomes: the TA acquires the units and completes the transaction resulting in a satisfied CA and a profit or loss for the TA, or the TA is unable or unwilling to purchase all of the units, resulting in an aborted transaction and the invocation of the penalty (which can involve both a monetary and a reputation component).

3. AGENT COMMUNICATION

3.1 Agent Communication Model

The previous TACs used a straightforward client-server architecture in which a single TAC server managed all travel service suppliers as well as the customers. Game participants wrote travel agency (TA) agents that connected as clients to the central TAC server. Moreover, these TA agents could only interact with service providers through centralized auction markets. While this architecture greatly simplifies both the development of the TAC infrastructure and the programming of a TAC client, it is a poor model for commerce in the real world.

Peer-to-peer or multi-agent systems offer a more realistic model where customers, service providers and various kinds of "middlemen", including market providers, operate as autonomous peer agents. Moreover, agents can develop complex strategies, which involved a combination of direct transactions (e.g., TA buy

direct from hotel agent) as well as auction-mediated transactions of various kinds. Finally, adopting a multi-agent systems approach integrated all aspects of commerce (service discovery, information seeking, negotiation, decision making, commitment, transaction execution et.) in a more natural manner.

The FIPA standards offer mature specifications for multi-agent systems communication, interactions and infrastructure with an emphasis on agent communication languages (ACLs) and protocols. We found the FIPA framework to be a good one for TAGA when augmented with the semantic web languages RDF [zou, 2003] and OWL. In the remainder of this section we will describe the choices made for the content languages.

3.2 OWL as Content Language

The content language is a language used to express the content of messages exchanged between agents. The FIPA communication infrastructure allows agents to communicate using any mutually understandable content language as long as it satisfies a few minimal criteria as a FIPA compliant content language. Published FIPA specifications provide a library of registered FIPA compliant content language, including FIPA-SL, XML and RDF. A good content language should be able to express rich forms of content and can be efficiently processed and fit well with existing technology. XML, used by the TAC system, is adequate as a low level language for encoding information but falls short as a language in which to express information at the knowledge level, even when augmented by more recent components such as XML Schema, XSL or through applications such as WSDL.

Our TAGA system uses OWL [Dean, 2002] as the content language for agent communication. Compared with RDF that used on our previous TAGA work [Zou, 2003], OWL has a well-defined model-theoretic semantics as well as an axiomatic specification that determines the intended interpretations of the language. OWL is unambiguously computer-interpretable, thus making it amenable to agent interoperability and automated reasoning techniques. The benefit of adopting a stronger semantically rich content language like OWL is that it facilitates a higher-level of interoperability between agents. By agreeing on how meaning is conveyed, it is simpler for applications to share meaningful content.

We have defined the OWL ontology for use as a FIPA-compliant content language. In addition to the basic required classes (e.g., Agent, ACLMessage, Service, etc.) and necessary expressive requirement (such as Proposition, Action, and Reification), our ontology provides supports for expressing rules, queries and responses to queries. We believe that OWL is a good choice as a general ACL content language for four reasons. First, its expressive power as a knowledge representation language seems to be adequate for many if not most needs of current agent based systems. Second, it offers better support for using terms drawn from multiple ontologies than do current popular ACL content languages. Third, as a semantic web language, it is designed to fit into and integrate with web-based information and service systems. Fourth, OWL has the potential to be a widely accepted and used representation language, enhancing the potential for interoperability among many systems. We will touch briefly on the first two points and leave the others as exercises for the reader.

To demonstrate that OWL is an adequate language for ACL content we consider a list of test cases presented in [Bothelo 2002]. These examples were used as an expressive test for a candidate FIPA content language and compared the result of encoding these in SL, KIF, ebXML, Prolog and DAML. Clearly OWL is less expressive than SL, KIF or Prolog, but the OWL version of these test cases given in Table 1 show that it's up to most of tasks it might be asked to serve.

Table 1: OWL Expressivity Test

Expression	Representation	Comment
"Schröding er's Cat is alive"	<Cat rdf:ID="schrödinger-s_cat"> <owner>Shrodinger</owner> <status> alive </status> </Cat>	There is a live cat in the world whose owner is Shrodinger.
"Cats are animals"	<owl:Class rdf:ID="cat"> <rdfs:subClassOf rdf:resource="#animal"> </owl:Class>	Cat is subclass of animal
"You making the tea"	<fipaowl:Action rdf:ID="tea_action1"> <fipaowl:act>making-tea </fipaowl:act> <fipaowl:actor>you</fipaowl:actor> <fipaowl:Action>	There is a making-tea action, "you" are the actor.
"Drinking too much is bad for you"	<Behavior rdf:ID="drinktoomuch"> <hasBehavior>excessive_drinking</ hasBehavior> <healthy>bad</healthy> </Behavior >	The behavior of drinking too much is bad for your health.
"All red things"	<owl:Class rdf:ID="allredthing "> <owl:intersectionOf rdf:parseType="Collection"> <owl:Classrdf:about="#Thing"/> <owl:Restriction> <owl:onProperty rdf:resource="#hasColor" /> <owl:hasValue rdf:resource="#Red" /> </owl:Restriction> </owl:intersectionOf> </owl:Class>	The things whose color are red.
"Any color a car might have"	<owl:Class rdf:ID="anycarcolor"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#color" /> <owl:allValuesFrom rdf:resource="#CarColor " /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	The color that limits the color property value in the car colors. This can also be a query: "Select color where color in Car Color"
"All things are hot"	<owl:Class rdf:about= "#Thing"> <rdfs:subClassOf>	All things's temperature

	<owl:Restriction> <owl:onProperty rdf:resource="#tempterature"/> <owl:hasValue rdf:resource="#hot" /> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	are hot.
"Somethin g is cold"	<owl:Thing rdf:ID= "cold_thing"> <temperature>cold</temperature> </owl:Thing>	There exist something whose temperature is cold.
"Herring or Perch"	<owl:oneOf rdf:parseType="Collection"> <owl:Thing rdf:about="#Vokda "/> <owl:Thing rdf:about="#Perch"/> </owl:oneOf>	
"Vodka and Tonic".	<owl:union Of rdf:parseType="Collection"> <owl:Class rdf:about="#Vodka " /> <owl:Class rdf:about="#Tonic" /> </owl:unionOf>	
"Not cricket"	<owl:Class rdf:ID="Noncricket"> <owl:complementOf rdf:resource="#Cricket " /> </owl:Class>	
"Success implies Payment"	<fipaowl:Rule> <fipaowl:Implies > <fipaowl:head > Payment</fipaowl:head > <fipaowl:body >Success </fipaowl:body > </fipaowl:Implies> </fipaowl:Rule>	The rule : Payment :- Success.
"Luis has the persistent goal that W"	<Person rdf:ID= "Luis"> <hasPersistentGoals> W </hasPersistentGoals> </Person>	
"Steve Believes X"	<Person rdf:ID="steve"> < hasProposition> < Belief rdf:ID="stevebelief1"> < believe>true</believe> < Statement > X</Statement> </Belief> </hasProposition> </Person >	
"Jonathan Desires Y"	<Person rdf:ID="Jonathan"> <hasProposition> <Desire rdf:ID="jonthandesire11"> <desire >true</desire> <Statement > Y </Statement> </Desire> </hasProposition> </Person >	
"Matthias Intends Z"	<Person rdf:ID="Matthias"> < hasProposition> <Intend rdf:ID="Matthiasintend1"> <intend>true</intend> <Statement > Z </Statement> </Intend> </hasProposition> </Person >	

Compared with other ACL content languages, OWL provides much better support in modeling, maintaining, and sharing ontologies. Standard content languages such as SL and KIF offer no explicit mechanisms for ontology support. FIPA inherited the simple mechanism for ontology specification first used in KQML that essentially required that all content terms in a particular message be tagged as coming from a single ontology. Although variations and "work arounds" to this constraint have been proposed, implemented and used, none have been formally adopted as part of the stable FIPA specification. OWL supports multiple namespaces and ontologies and, in fact, is a large part of its raison d'etre. Large scale and open multi-agent systems will benefit from OWL's abilities to integrate information from different ontologies. Moreover, OWL and other semantic web languages, will better support other services essential to large scale open systems, such as the capability to translate or map information from one ontology to another and to negotiate meaning or otherwise resolve differences between ontologies.

3.3 Understanding Messages

When an agent receives an incoming ACL message, it computes the meaning of the message from the ACL semantics, the protocols in effect, the content language and the conversational context. The agent's subsequent behavior, both internal (e.g., updating its knowledge base) and external (e.g., generating a response) depends on the correct interpretation of the message's meaning. Thus, a sound and, if possible, complete understanding the *semantics* of the key communication components (ACL, protocol, ontologies, content language, context) is extremely important. In TAGA, the service providers are independent and autonomous entities, which makes enforcing a design decision that all use exactly the same ontology or protocol difficult, if not impossible. For example, the Delta Airline service agent may has its own view of travel business and uses class and property terms that extend an ontology used in the industry as a whole. This situation parallels that for the semantic web as a whole – some amount of diversity is inevitable and must be panned for lest our systems become impossibly brittle.

The ontologies in TAGA are distributed and managed by multiple parties. This distributed model is a better fit for deployment in an open web environment. There is no centralized site or agent that has to understand every ontologies. Ontologies and rules are designed and implemented by service owners to reflect their business models and meet their requirements; tan agent belonging to a service owner is responsible for answering the question related to the ontologies it uses. Ontologies store in local and may access only by local agent. We could define personalized ontologies and rules. It would help resolving the problem of security and trust.

Many of the agents we have implemented in the TAGA system use FOWL (Flora OWL) to represent and reason about content presented in RDF or OWL. FOWL is a flora-2 [Yang 2000] program that interprets RDF and OWL represented as a collection of RDF triples. Flora-2 is itself a compiler that compiles from a dialect of f-logic into XSB, taking advantage of the tabling, HiLog and well-founded semantics for negation features found in XSB. On receiving an ACL message with content in RDF or OWL, a TAGA agent parses the content into triples, which are then loaded into the XSB engine for processing.

The message's meaning (communicative act, protocol, content language, ontologies and context) all play a part in the interpretation. For example, receiving a query message using query protocol, the agent searches its knowledge base for matching answers and returns an appropriate inform message. TAGA uses multiple models to reflect the multiple namespace and ontologies in the system. The agent treats each ontology as an independent Model in XSB engine. The support of ontology sharing and exchanging is achieved by defining a set of ontology related actions:

- *NewInstance:* this message creates an instance using the specified ontology and the provided instance data;
- *OntologyQuery:* this message queries other agents about the terms defined in their ontology;
- *OntologyShare:* this inform message is about the ontology definition, which include Class/Property definition, Class-Subclass relation and Class-Property relation.
- *OntologyRelation:* this message is about the conversion and relations among class or property term defined different ontologies. For example, agent A informs agent B that the class Person is same class as the class Human used by agent B. The relations include extension, identical and equivalent. This message can be an inform message informing other agents about the relation, or query message asking to confirm the relation, or request message asking to translate the ontology term used in multiple ontologies.

3.4 Query Support

Among the most important communicative acts used by agents are those designed to support querying. The FIPA ACL has a very simple query model supporting just two acts -- *query-if* and *query-ref* – but allows a more complicated query to be encoded as a *request* act. In order to use semantic web languages for ACL content, we have experimented with the integration of a number of RDF based approaches, including DQL, RQL, RDQL, Triple, and TAP. Since a consensus query system has not yet emerged, we have adopted an approach in which agents can use any of several query systems and associated protocols. An agent specifies the query languages and protocols it understands as part of its basic service description. Other agents who intend to submit query to this agent are expected to encode the query string in one of the support languages. Table 2 is a query and answer example using RDQL language.

Table 2: Query and answer example

Query:

<fipaowl:Query rdf:ID="query1">

<fipaowl:queryLanguage>rdql</fipaowl:queryLanguage>

<fipaowl:question>

"SELECT ?x,?y

FROM <people.rdf>

WHERE (?x,<dt:friend>,?y),(?y,<dt:friend>,?x)

AND ?x<^gt;?y

USING dt for <http://foo.org#>, rdf for http://www.w3.org/1999/02/22-rdf-syntax-ns#"

</fipaowl:question>

<fipaowl:result_number>10</fipaowl:result_number>

</fipaowl:Query >

Answer:

<fipaowl:Query rdf:about="query1">

<fipaowl:queryLanguage>rdql</fipaowl:queryLanguage>

<fipaowl:result_number>1</fipaowl:result_number>

<fipaowl:answer>

"Array ([0] => Array ([?x] => http://foo.org/persons/Carl [?y] => http://foo.org/persons/Peter) [1] => Array ([?x] => http://foo.org/persons/Peter [?y] => http://foo.org/persons/Carl))"

</fipaowl:answer>

</fipaowl:Query>

We have found that the basic framework of FIPA standards support this approach well by having a good set of primitive communicative acts, a way for agents to define communication protocols, and a sound mechanism by which agents can describe their capabilities and the supporting services. We are planning to experiment with adding mediator agents to TAGA that offer a query translation service. Such an agent would be able to handle several kinds of query languages permitting it to act as a proxy. For example, agent A might wish to ask a DQL query of agent B, which only understands RQL. A query translation service able to process both DQL and RQL could provide the mediation service – receiving a DQL query from A, sending appropriate RQL queries to B, accepting the response, and reformulating to fit the DQL protocol.

4. DISCUSSION

In this section we will briefly discuss several additional design issues we have addressed in TAGA.

Ontologies. In addition to the FIPA content language ontology, we have defined two domain ontologies in OWL. The first is a travel ontology that covers the basic concepts of traveling needed in TAGA, include the travel itinerary, customers, travel services and service reservations. The second ontology is one for auctions. This ontology is used to define the different kinds of auctions, the roles the participants play in them, and the protocols used.

Service description and matching. FIPA agents are associated with one or more FIPA platforms, each of which offers a set of agent services including a Directory Facility (DF) agent that handles service registration, deregistration and matching. When an agent registers a service in a DF, it provides service information like the service type and owner. However, more specific service information may also be useful when searching for agent services. For example, a customer may want a booking in a hotel with at least three star rating, is close to public transportation, offers breakfast, and accepts VISA card payments. This can be achieved with the use of DAML-S [DAML-S, 2002] profile. In TAGA, every travel service provider describes its service process model with DAML-S language and publishes it as a web page. This covers basic service information like address, phone number and service interface information. For example, a hotel may describe booking service as: customer name, payment methods, travel date as input; reserve

number as output; the effect of booking is one room occupied at the travel date. The travel agent, who is responsible for organizing travel package, is able to contact with customer agent and related service agents and finds the best match. First the travel agent loads the DAML-S parsing rule and planning rules into its XSB reasoning engine. It then loads service agents' DAML-S profiles and customer's personal profile. The best matching service providers are selected and a most profitable travel package is composed dynamically.

Implementation comments. The original Trading Agent Competitions relied on a few centralized market servers to handle all interactions and coordination, including service discovery, agent communication, coordination, and game control. In contrast, the TAGA framework uses a distributed peer-to-peer approach based on standard agent languages, protocols and infrastructure components (FIPA, Agentcities), emerging standards for representing ontologies, knowledge and services (RDF, OWL, DAML-S) and web infrastructure (e.g., Sun's Java Web Start). Several FIPA platform implementations are currently used within TAGA, including Jade and AAP (April Agent Platform), demonstrating agent interoperability. Our current demonstration system allows new users to dynamically join a running game at any time. A dummy agent implemented in JADE can be downloaded and run to instantiate a new TA agent. A simple GUI allows the user to set operating parameters or the java code can be modified or extended. A set of web based monitoring services allow one to see the status of a game, examine messages being sent, lookup the reputation of agents, etc.

Contribution. We see two main contributions in our work. First, TAGA provides a rich framework for exploring agent-based approaches to e-commerce like applications. Our current framework allows users to create their own agent (perhaps based on our initial prototype) to represent a TA or SA and to include it in a running game where it will compete with other system provided and user defined agents. We hope that this might be a useful teaching and learning tool, not only for multi-agent systems technology, but also for the semantic web languages RDF and OWL and their use in agent based systems. Secondly, we hope that TAGA will be seen as a flexible, interesting and rich environment for simulating agent-based trading in dynamic markets. Agents can be instantiated to represent customers, aggregators, wholesalers, and service provides all of which can make decisions about price and purchase strategies based on complex strategies and market conditions. Moreover, simulations like TAGA encourage exploring aspects of e-commerce that go beyond auction theory. TA agents might compete on their ability to better understand the descriptions of services sought and services offered and the basic models of the preferences of their users in order to best satisfy the needs of their clients. These descriptions, of course, will be in a semantic web language like OWL.

5. Conclusions and future work

Travel Agent Game in Agentcities (TAGA) is a framework that extends and enhances the Trading Agent Competition (TAC) system to work in Agentcities, an open multiagent systems environment of FIPA compliant systems. We hope that TAGA will serve as an experimental test-bed for several communities of users.

First, it provides an environment, which can be used to explore aspects of multiagent systems technology based on the mature, published FIPA standards. Research on multiagent systems technology is best done with in a rich yet easily understood problem domain. We have found that the travel agent scenario as originally put forth by TAC provides both the richness as well as accessibility, especially when opened up to be peer-to-peer. We are using TAGA as a test-bed for research on the use of semantic web languages (e.g., RDF and OWL) as content languages and as service description languages. Future work is planned in adding more sophisticated negotiation and ontology mapping to our TAGA environment.

Second, we hope that TAGA could serve as an interesting framework and test-bed for experiments with automated markets and trading. By adding autonomous service provide agents (e.g., for hotels) one could experiment with a dynamic market with both "shopbots" and "pricebots" or investigate the role of intermediation in the form of agents performing a wholesale function.

Third, we hope that others will find TAGA useful as a test, demonstration and teaching environment, both in technology classes focused multi-agent systems, FIPA standards or the semantic web and in business or e-commerce classes focused on automating commerce and trading, auctions or agent-based simulations.

The Agentcities project is exploring the delivery and use of agent-based services in an open, dynamic and international setting. We are working to increase the integration of TAGA and emerging Agentcities components and infrastructure and will include agents running on handheld devices using LEAP.

6. REFERENCES

[Bothelo 2002] L. Bothelo, S. Willmott, T. Zhang, J. Dale, A review of Content Languages Suitable for Agent-Agent Communication, EPFL I&C Technical Report #200233.

[Dale, 2002] J. Dale, S. Willmot, and B.Burg: Agentcities: Challenges and Deployment of Next-Generation Service Environments. Proc. Pacific Rim Intelligent Multi-Agent Systems, Tokyo, Japan, August 2002.

[DAML-S, 2002] The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara): DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.

[Dean, 2002] M. Dean and Guus Schreiber (eds): OWL Web Ontology Language 1.0 Reference. W3C Working Draft.

[Greenwald, 2001] Amy Greenwald and Peter Stone: Autonomous Bidding Agents in the Trading Agent Competition, IEEE Internet Computing, March/April 2001.

[Greenwald, 2003] Amy Greenwald (ed.). The 2002 trading agent competition: An overview of agent strategies. AI Magazine, to appear

[Sagonas, 1994] Kostantinos Sagonas, Terrance Swift, and David S. Warren: XSB as an efficient deductive database engine, In ACM Conference on Management of Data (SIGMOD), 1994.

[Stone, 2000] Peter Stone and Amy Greenwald: The First International Trading Agent Competition: Autonomous Bidding Agents, Electronic Commerce Research Journal pp1-36, 2000.

[Wellman, 2002] Michael P. Wellman, Amy Greenwald, Peter Stone, and Peter R. Wurman: The 2001 Trading Agent Competition, Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-2002), pp935-941, Edmonton, August 2002.

[Willmott, 2001] Willmott, S., Dale, J., Burg, B., Charlton, P. and O'Brien, P., Agentcities: A Worldwide Open Agent Network. In: AgentLink News, Issue 8, November 2001.

[Yang, 2000] Guizhen Yang and Michael Kifer. FLORA: Implementing an efficient DOOD system using a tabling logic engine. Proceedings of Computational Logic --- CL-2000, number 1861 in LNAI, pp 1078--1093. Springer, July 2000.

[Zou, 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan, TAGA: Trading Agent Competition in Agentcities, Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, Monday, 11 August, 2003, Acuulco MX.

F-OWL: an Inference Engine for the Semantic Web [1]

Youyong Zou, Tim Finin and Harry Chen

Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore MD 21250
{yzou1,finin, hchen4 }@cs.umbc.edu

Abstract. Understanding and using the data and knowledge encoded in semantic web documents requires an inference engine. F-OWL is an inference engine for the semantic web language OWL language based on F-logic, an approach to defining frame-based systems in logic. F-OWL is implemented using XSB and Flora-2 and takes full advantage of their features. We describe how F-OWL computes ontology entailment and compare it with other description logic based approaches. We also describe TAGA, a trading agent environment that we have used as a test bed for F-OWL and to explore how multiagent systems can use semantic web concepts and technology.

1 Introduction

The central idea of the Semantic Web [Berners-Lee 2001] is to publish documents on the World Wide Web defined and linked in a way that make them both human readable and machine understandable. Human readable means documents in the traditional sense which are intended for machine display and human consumption. Machine understandable means that the data has explicitly been prepared for machine reasoning and reuse across various applications. Realizing the semantic web vision requires well defined languages that can model the meaning of information on the Web as well as applications and services to publish, discover, process and annotate information encoded in them. This involves aspects from many areas, including knowledge representation and reasoning, databases, information retrieval, digital libraries, multi-agent systems, natural language processing and machine learning. The Web Ontology Language OWL [Patel-Schneider, 2003] is part of the growing stack of W3C recommendations related to the Semantic Web. OWL has its origins in DAML+OIL [Hendler 2000] and includes a set of three increasingly complex sub-languages: OWL-Lite, OWL-DL and OWL-Full.

OWL has a model-theoretic semantics that provides a formal meaning for OWL ontologies and instance data expressed in them. In addition, to support OWL-Full, a

[1] This work was partially supported by the Defense Advanced Research Projects Agency under contract F30602-97-1-0215 and by the National Science Foundation under award IIS-0242403.

second model-theoretic semantics has been developed as an extension to the RDF's semantics, grounding the meaning of OWL ontologies as RDF graphs. An OWL inference engine's core responsibilities are to adhere to the formal semantics in processing information encoded in OWL, to discover possible inconsistencies in OWL data, and to derive new information from known information. A simple example demonstrates the power of inference: Joe is visiting San Francisco and wants to find an Italian restaurant in his vicinity. His wireless PDA tries to satisfy his desire by searching for a thing of type *restaurant* with a *cuisineType* property with the value *Italian*. The goodPizza restaurant advertises its cuisine type as *Pizza*. These cannot be matched as keywords or even using a thesaurus, since *Italian* and *Pizza* are not equivalent in all contexts. The restaurant ontology makes things clearer: *Pizza rdfs:SubClassOf ItalianCuisine*. By using an inference engine, Joe's PDA can successfully determine that the restaurant goodPizza is what he is looking for. F-OWL, an inference engine for OWL language, is designed to accomplish this task.

In the next section, we outline the functional requirement of the OWL inference engine. Section three describes F-OWL, the OWL inference engine in Frame Logic that we have developed. Section four explained how F-OWL is used in a multi-agent test bed for trading agents. Chapters five and six conclude this paper with a discussion of the work and results and an outline of some potential future research.

2 OWL Engine

An inference engine is needed for the processing of the knowledge encoded in the semantic web language OWL. An OWL inference engine should have following features:

- **Checking ontology consistency.** An OWL concept ontology (e.g., terms defined in the "Tbox") imposes a set of restrictions on the model graph. The OWL inference Engine should check the syntax and usage of the OWL terms and ensure that the OWL instances (e.g., assertions in the "Abox") meet all of the restrictions.
- **Computing entailments.** Entailment, including satisfiability and subsumption, are essential inference tasks for an OWL inference engine.
- **Processing queries.** OWL inference engines need powerful, yet easy-to-use, language to support queries, both from human users (e.g., for debugging) and software components (e.g., for software agents).
- **Reasoning with rules**. Rules can be used to control the inference capability, to describe business contracts, or to express complex constrictions and relations not directly supported by OWL. An OWL inference engine should provide a convenient interface to process rules that involve OWL classes, properties and instance data.
- **Handling XML data types.** XML data types can be used directly in OWL to represent primitive kinds of data types, such as integers, floating point numbers, strings and dates. New complex types can be defined using base types

and other complex types. An OWL inference Engine must be able to test the satisfiability of conjunctions of such constructed data types.

The OWL language is rooted in description logic (DL), a family of knowledge representation languages designed for encoding knowledge about concepts and concept hierarchies. Description Logics are generally given a semantics that make them subsets of first-order logic. Therefore, several different approaches based on those logics have been used to design OWL inference engines:

- **Using a specialized description logic reasoner.** Since OWL is rooted in description logic, it is not surprising that DL reasoners are the most widely used tools for OWL reasoning. DL reasoners are used to specify the terminological hierarchy and support subsumption. It has the advantage of being decidable. Three well-known systems are FaCT [Horrocks, 1999], Racer [Haarslev 2001] and Pellet. They implement different types of description logic. Racer system implements SHIQ(D) using a Tableaux algorithm. It is a complete reasoner for OWL-DL and supports both Tbox and Abox reasoning. The FaCT system implements SHIQ, but only support Tbox reasoning. Pellet implements SHIN(D) and includes a complete OWL-lite consistency checker supporting both Abox and Tbox queries.
- **Using full first order logic (FOL) theorem prover.** OWL statements can be easily translated into FOL, enabling one to use existing FOL automated theorem provers to do the inference. Examples of this approach include Hoolet (using the Vampire [Riazanov, 2003] theorem prover) and Surnia (using Otter theorem prover). In Hoolet, for example, OWL statements are translated into a collection of axioms which is then given to the Vampire theorem prover for reasoning.
- **Using a reasoner designed for a FOL subset.** A fragment of FOL and general logic based inference engine can also be used to design the OWL inference engine. Horn Logic is most-widely used because of its simplicity and availability of tools, including Jena, Jess, Triple and F-OWL (using XSB). Other logics, like higher-order logic in F-OWL (using Flora), can also be used.

As the following sections describe, F-OWL has taken the third approach. An obvious advantage is that many systems have been developed that efficiently reason over expressive subsets of FOL and are easy to understand and use.

3 F-OWL

F-OWL is a reasoning system for RDF and OWL that is implemented using the XSB logic programming system [Sagonas, 1994] and the Flora-2 [Kifer, 1995] [Yang 2000] extension that provides an F-logic frame-based representation layer. We have found that XSB and Flora-2 not only provide a good foundation in which to implement an OWL reasoner but also facilitate the integration of other reasoning mechanisms and applications, such as default reasoning and planners.

XSB is a logic programming system developed at Stony Brook University. In addition to providing all the functionality of Prolog, XSB contains several features not usually found in Logic Programming systems, including tabling, non-stratified negation, higher order constructs, and a flexible preprocessing system. Tabling is useful for recursive query computation, allowing programs to terminate correctly in many cases where Prolog does not. This allows, for example, one to include "if and only if" type rules directly. XSB supports for extensions of normal logic programs through preprocessing libraries including a sophisticated object-oriented interface called Flora-2. Flora-2 is itself a compiler that compiles from a dialect of Frame logic into XSB, taking advantage of the tabling, HiLog [Chen 1995] and well-founded semantics for negation features found in XSB. Flora-2 is implemented as a set of run-time libraries and a compiler that translates a united language of F-logic and HiLog into tabled Prolog code. HiLog is the default syntax that Flora-2 uses to represent function terms and predicates. Flora-2 is a sophisticated object-oriented knowledge base language and application development platform. The programming language supported by Flora-2 is a dialect of F-logic with numerous extensions, which include a natural way to do meta-programming in the style of HiLog and logical updates in the style of Transaction Logic. Flora-2 was designed with extensibility and flexibility in mind, and it provides strong support for modular software design through its unique feature of dynamic modules.

F-OWL is the OWL inference engine that uses a Frame-based System to reason with OWL ontologies. F-OWL is accompanied by a simple OWL importer that reads an OWL ontology from a URI and extracts RDF triples out of the ontology. The extracted RDF triples are converted to format appropriate for F-OWL's frame style and fed into the F-OWL engine. It then uses flora rules defined in flora-2 language to check the consistency of the ontology and extract hidden knowledge via resolution.

A model theory is a formal theory that relates expressions to interpretation. The RDF model theory [Hayes 2003] formalizes the notion of inference in RDF and provides a basis for computing deductive closure of RDF graphs. The semantics of OWL, an extension of RDF semantics, defines bindings, extensions of OWL interpretations that map variables to elements of the domain:

- The vocabulary *V* of the model is composed of a set of **URI**'s.
- *LV* is the set of *literal values* and *XL* is the mapping from the literals to *LV*.
- A *simple interpretation I* of a vocabulary *V* is defined by:
  - A non-empty set *IR* of resources, called the domain or universe of *I*.
  - A mapping *IS* from *V* into *IR*
  - A mapping *IEXT* from *IR* into the power set of *IR X (IR union LV)* i.e. the set of sets of pairs *<x,y>* with *x* in *IR* and *y* in *IR* or *LV*. This mapping defines the properties of the triples. *IEXT(x)* is a set of pairs which identify the arguments for which the property is true, i.e. a binary relational extension, called the *extension* of x.

Informally this means that every **URI**[2] represents a resource that might be a page on the Internet but not necessarily; it might also be a physical object. A property is a relation; this relation is defined by an extension mapping from the property into a set. This set contains pairs where the first element of a pair represents the subject of a triple and the second element represents the object of a triple. With this system of extension mapping the property can be part of its own extension without causing paradoxes.

Take the triple:*goodPizza :cuisineType :Pizza* from the pizza restaurant in the introduction as example. In the set of **URI**'s there will be terms (i.e., classes and properties) like: *#goodPizza, #cuisineType, #pizza, #Restanrant, #italianCuisine,* etc. These are part of the vocabulary *V*. The set **IR** of resources include instances that represent resources on the internet or elsewhere, like *#goodPizza*, , etc. For example the class *#Restanrant* might represent the set of all restaurants. The **URI** refers to a page on the Internet where the domain **IR** is defined. Then there is the mapping *IEXT* from the property *#cuisineType* to the set {(*#goodPizza, #Pizza),(#goodPizza, #ItalianCuisine)*} and the mapping *IS* from *V* to *IR*: *:goodPizza → #goodPizza, :cuisineTYpe → #cuisineType.*

A rule *A→B* is satisfied by an interpretation *I* if and only if every binding that satisfies the antecedent *A* also satisfies the consequent *B.* An ontology *O* is satisfied by an interpretation *I* if and only if the interpretation satisfies every rules and facts in the ontology. A model is satisfied if none of the statements within contradict each other. An ontology O is consistent if and only if it is satisfied by at least one interpretation. An ontology O_2 is entailed by an ontology O_1 if and only if every interpretation that satisfies O_1 also satisfies O_2.

One of the main problems in OWL reasoning is ontology entailment. Many OWL reasoning engines, such as Pellet and SHOQ, follow an approach suggested by Ian Horrocks [Horrocks 2003]. By taking advantage of the close similarly between OWL and description logic, the OWL entailment can be reduced to knowledge base satisfiability in the SHOIN(D) and SHIF(D). Consequently, existing mature DL reasoning engines such as Racer [Haarslev 2001] can provide reasoning services to OWL. Ora Lassila suggested a *"True RDF processor"* [Lassila 2002] in his implementation of Wilbur system [Lassila 2001] in which entailment is defined via the generation of a deductive closure from an RDF graph composed of triples. The proving of entailment becomes the building and searching of closure graph.

With the support of forward/backward reasoning from XSB and frame logic from Flora, F-OWL takes the second approach to compute the deductive closure of a set of RDF or OWL statements. The closure is a graph consisting of every triples <*subject, predicate, object*> that satisfies *{subject, object } ⇒ IEXT(I(predicate)).* This is defined as:

$$<subject,predicate,object> \Rightarrow KB \Leftrightarrow \{subject,object\} \Rightarrow IEXT(I(predicate))$$

[2] The W3C says of URIs: "Uniform Resource Identifiers (URIs, aka URLs) are short strings that identify resources in the web: documents, images, downloadable files, services, electronic mailboxes, and other resources." By convention, people understand many URIs as denoting objects in the physical world.

Where *KB* is the knowledge base, *I(x)* is the interpretation of a particular graph, and *IEXT(x)* is the binary relational extension of property as defined in [Hayes 2002].

F-OWL is written in the Flora-2 extension to XSB and consists of the following major sets of rules:

- A set of rules that reasons over the data model of RDF/RDF-S and OWL;
- A set of rules that maps XML DataTypes into XSB terms;
- A set of rules that performs ontology consistency checks; and
- A set of rules that provides an interface between the upper Java API calls to the lower layer Flora-2/XSB rules.

F-OWL provides command line interface, a simple graphical user interface and a Java API to satisfy different requirements. Using F-OWL to reason over the ontology typically consists of the following four steps:

- Loading additional application-related rules into the engine;
- Adding new RDF and OWL statements (e.g., ontologies or assertions) to the engine. The triples (subject, predicate, object) on the OWL statements are translated into 2-ply frame style: subject(predicate, object)@model;
- Querying the engine. The RDF and OWL rules are recursively applied to generate all legal triples. If a query has no variables, a True answer is returned when an interpretation of the question is found. If the question includes variable, the variables is replaced with values from the interpretation and returned;
- The ontology and triples can be removed if desired. Else, the XSB system saves the computed triples in indexed tables, making subsequent queries faster.

4 F-OWL in TAGA

Travel Agent Game in Agentcities (TAGA) [Zou 2003] is a travel market game developed on the foundation of FIPA technology and the Agentcities infrastructure. One of its goals is to explore and demonstrate how agent and semantic web technology can support one another and work together.

TAGA extends and enhances the Trading Agent Competition scenario to work in Agentcities, an open multiagent systems environment of FIPA compliant systems. TAGA makes several contributions: auction services are added to enrich the Agentcities environment, the use of the semantic web languages RDF and OWL improve the interoperability among agents, and the OWL-S ontology is employed to support service registration, discovery and invocation. The FIPA and Agentcities standards for agent communication, infrastructure and services provide an important foundation in building this distributed and open market framework. TAGA is intended as a platform for research in multiagent systems, the semantic web and/or automated trading in dynamic markets as well as a self contained application for teaching and experimentation with these technologies. It is running as a continuous open game at

http://taga.umbc.edu/ and source code is available on Sourceforge for research and teaching purposes.

The agents in TAGA use OWL in various ways in communication using the FIPA agent content language (ACL) and also use OWL-S as the service description language in FIPA's directory facilitators. Many of the agents in the TAGA system use F-OWL directly to represent and reason about content presented in OWL. On receiving an ACL message with content encoded in OWL, a TAGA agent parses the content into triples, which are then loaded into the F-OWL engine for processing.

When an agent receives an incoming ACL message, it computes the meaning of the message from the ACL semantics, the protocols in effect, the content language and the conversational context. The agent's subsequent behavior, both internal (e.g., updating its knowledge base) and external (e.g., generating a response) depends on the correct interpretation of the message's meaning. Thus, a sound and, if possible, complete understanding the *semantics* of the key communication components (i.e., ACL, protocol, ontologies, content language, context) is extremely important. In TAGA, the service providers are independent and autonomous entities, which making it difficult to enforce a design decision that all use exactly the same ontology or protocol. For example, the Delta Airline service agent may have its own view of travel business and uses class and property terms that extend an ontology used in the industry. This situation parallels that for the semantic web as a whole – some amount of diversity is inevitable and must be panned for lest our systems become impossibly brittle.

Many of the agents implemented in TAGA system use F-OWL to represent and reason about the message content presented in RDF or OWL. Upon receiving an ACL message with content in RDF or OWL, a TAGA agent parses the content into triples, which are then loaded into the FOWL engine for processing.

The message's meaning (communicative act, protocol, content language, ontologies and context) all play a part in the interpretation. For example, when an agent receives a query message that uses the query protocol, the agent searches its knowledge base for matching answers and returns an appropriate inform message. TAGA uses multiple models to reflect the multiple namespaces and ontologies used in the system. The agent treats each ontology as an independent model in the F-OWL engine.

F-OWL has many usages in TAGA, including the following.

- **As knowledge base.** Upon receiving an ACL message with content encoded in OWL, agents in TAGA parse the content into triples and feeds them into their F-OWL engine. The information can be easily retrieved by submitting queries in various query languages.
- **As reasoning engine.** The agent can answer more questions with the help of F-OWL engine, for example, the restaurant can answer the question "what is the average price of a starter" after it understands that "starter" is *sameAs* "appetizer".
- **As a service matchmaker.** FIPA platforms provide a *directory facilitator* service which matches service requests against descriptions of registered services. We have extended this model by using OWL-S as a service description language.

F-OWL manages the service profiles and tries to find the best match based on description in the service request.

- **As an agent interaction coordinator.** The interaction protocol can be encoded into an ontology file using OWL language. F-OWL will advise the agents what to respond based on received messages and context.

5 Discussion

This section describes the design and implementation of F-OWL, an inference engine for OWL language. F-OWL uses a Frame-based System to reason with OWL ontologies. F-OWL supports consistency checking of the knowledge base, extracts hidden knowledge via resolution and supports further complex reasoning by importing rules. Based on our experience in using F-OWL in several projects, we found it to be a fully functional inference engine that was relatively easy to use and able to integrate with multiple query languages and rule languages.

There have been lots of works on the OWL inference engine, from semantic web research community and description logic community. The following table compares F-OWL with some of them:

Table 1: Comparison of F-OWL and other OWL Inference Engine

	F-OWL	**Racer**	**FaCT**	**Pellet**	**Hoolet**	**Surnia**	**Triple**
Logic	Horn, Frame, Higher Order	Description Logic	DL	DL	Full FOL	Full FOL	Horn Logic
Support	OWL-Full	OWL-DL	OWL-DL	OWL-DL	OWL-DL	OWL-Full	RDF
Based on	XSB/Flora	Lisp	Lisp	Java	Vampire	Otter	XSB
XML Datatype	Yes	Yes	No	Yes	No	No	No
Decidable	No	Yes	Yes	Yes	No	No	Yes
Complete consistency checker	No	Yes (OWL-Lite)	Yes	Yes(OWL-Lite)	No	No	No
Interface	Java, GUI, Command Line	DIG, Java, GUI	DIG, Command Line	DIG, Java	Java	Python	Java
Query	Frame style, RDQL	Racer query language		RDQL			Horn logic style
Known Limitation	Poor scaling		No Abox support		Poor scaling	Poor scaling	Only support RDF

The first thing to notice in Table 1 is that the description logic based system can only support reasoning over OWL-Lite and OWL-DL statements but not OWL-Full. OWL-Full is a full extension of RDF, which needs the supporting of terminological cycle. For example, a class in OWL-Full can also be an individual or property. The cyclic terminological definitions can be recognized and understood in horn logic or frame logic system.

Table 1 shows that only three DL-based owl inference engines, which are all use a Tableau based algorithms [Baader 2000], are decidable and support complete consistency checking (at least in OWL-Lite). However, [Balaban 1993] argues that DL only forms a subset of F-Logic. The three kinds of formulae in the description logic can be transformed into first class objects and n-ary relationships. F-Logic is able to provide a full account for DL without losing any semantics and descriptive nature. We understand that our current F-OWL approach is neither decidable nor complete. However, a complete F-Logic based OWL-DL reasoner is feasible.

The table also shows that F-OWL system doesn't scale well when dealing with large datasets, because of the incompleteness of the reasoner. Actually, none of the OWL inference engines listed here scales well when dealing with the OWL test case wine ontology[3] which defines thousands of classes and properties and a relatively modest number of individuals. Further research is needed to improve the performance and desirability.

Comparing with other OWL inference engines, F-OWL has several unique features: tabling, support for multiple logical models or reasoning, and a pragmatic orientation.

Tabling. XSB's tabling mechanism gives F-OWL the benefits of a forward chaining system in a backward chaining environment. The triples in a model are computed only when the system needs to know whether or not they are in the model. Once it is established that a triple is in the current model, it is added to the appropriate table, obviating the need to prove that it is in the model again. This mechanism can have a significant impact on the system's performance. While the first few queries may take a long time, subsequent queries tend to be very fast. This is an interesting compromise between a typical forward-only reasoning system and backward-only reasoning systems.

Multiple logics. F-OWL supports Horn logic, frame logic and a kind of higher-order logic; all inherited from the underlying XSB and Flora substrates. Working together, these logic frameworks improve F-OWL's performance and capabilities. For example, the F-logic supports non-monotonic (default) reasoning. Another example is higher-order logic. The semantics of higher-order logics, in general, are difficult and in many cases not suitable for practical applications. XSB's Hilog, however, is a simple syntactic extension of first-order logic in which variables can appear in the position of a predicate. In many cases, this simplifies the expression of the statements, rules and constraints, improving the writability and readability of F-OWL and associated programs.

[3] The wine ontology is used as a running example in the W3C's OWL Web Ontology Language Guide and is available at http://www.w3.org/TR/owl-guide/wine.owl.

Pragmatic approach. The aim of F-OWL system is to be a practical OWL reasoner, not necessary a complete OWL reasoner. So F-OWL system provides various interface to access the engine and supports multiple query and rule languages.
In the open web environment, it is generally assumed that the data are not complete and not all facts are known. We will research how this fact affects the implementation of inference engine. In the semantic web an inference engine may not necessarily serve to generate proofs but should be able to check proofs. We will work on using F-OWL to resolve trust and proof in semantic web.

In a stand-alone system inconsistencies are dangerous but can be controlled to a certain degree. However, controlling the inconsistencies in the Semantic Web is a lot more difficult. During the communication, ontology definition origin from other agents, who is unknown beforehand, may be asserted. Therefore special mechanisms are needed to deal with inconsistent and contradictory information in the Semantic Web. There are two steps: detecting the inconsistency and resolving the inconsistency.

The detection of the inconsistency is based on the declaration of inconsistency in the inference engine. The restriction, which imposes the possible values and relation that the ontology elements can have, leads to the inconsistency. For example, *owl:equivalentClass*: imposes a restriction on the resource which the subject is same class as. *owl:disjointWith* imposes a restriction on the resource which the subject is different from. The triples *(a owl:equivalentClass b)* and *(a owl:disjointWith b)* is not directly lead to an inconsistency until applying the detection rule: *(A owl:equivalentClass B) & (A owl:disjointWith B)* ➔*inconsistency.*

When inconsistencies are detected, Namespaces can help tracing the origin of the inconsistencies. John posted "all dogs are human" at his web site, while "all dogs are animal" appears in daml.org's ontology library. It is clear that the second is more trustable. Every web site are identified and treated unequivocally in the semantic web. The inference engine contacts trust system to evaluate the creditability of the namespaces. [Klyne 2002] and [Golbeck 2003] enlist lots of works and brilliant ideas about how to maintain the trust system in the semantic web. Once having the trust evaluation result, the agent could take three different actions: (a) accept the one suggested by the inference engine; (b) reject both as none of them is trustable; (c) ask the human user to select.

6 Conclusion

This paper describes the design and implementation of F-OWL, an inference engine for OWL language. F-OWL uses a Frame-based System to reason with OWL ontologies. F-OWL supports consistency checking, extracts hidden knowledge via resolution and supports further complex reasoning by importing rules. While using it in TAGA user case, we find that F-OWL is a full functional inference engine and easy to use with the support of multiple query languages and rule languages.

In the open web environment, it is generally assumed that the data are not complete and not all facts are known. We will research how this fact affects the implementation of inference engine. In the semantic web an inference engine may not nec-

essarily serve to generate proofs but should be able to check proofs. We will work on using F-OWL to resolve trust and proof in semantic web in the future.

References

[Baader 2000] Franz Baader and Ulrike Sattler: "An Overview of Tableau Algorithms for Description Logics", Proceeding of Tableau 2000, RWTH Achen.

[Balaban 1993] Mira Balaban: "The F-Logic Approach for Description Languages", Ben-Gurion University of Negev Technical Report FC-93-02, 1993.

[Berners-Lee 2001] Tim Berners-Lee, James Hendler and Ora Lassila: "The Semantic Web", Scientific America , May 2001.

[Chen 1995] Weidong Chen, Michael Kifer and David Warren: "Logical Foundations of Object-Oriented and Frame-Based Languages", Journal of ACM, May 1995.

[Golbeck 2003] Jennifer Golbeck, Bijan Parsia, and James Hendler: "Trust networks on the semantic web", Proceedings of Cooperative Intelligent Agents 2003, Helsinki, Finland, August 2003.

[Haarslev 2001] Volker Haarslev and Ralf Moller: "Racer system description", Proceeding of International Joint Conference on Automated Reasoning, Volume 2083, page 701-705, Springer 2001

[Hayes 2003] Pat Hayes, RDF Semantics , W3C working Draft, 2003.

[Hendler 2000] James Hendler and Deborah L. McGuinness, "The DARPA Agent Markup Language." IEEE Intelligent Systems, Trends and Controversies, page. 6-7, November/December 2000.

[Horrocks, 1999] I. Horrocks. FaCT and iFaCT. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133-135, 1999.

[Horrocks, 2003] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17-29. Springer, 2003.

[Kifer, 1995] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. JACM, 42(4):741--843, Jul. 1995.

[Klyne 2002] G.Klyne: "Framework for Security and Trust Standards", SWAD-Europe, December 2002.

[Lassila 2001] Ora Lassila: Enabling Semantic Web Programming by Integrating RDF and Common lisp", Proceeding of first Semantic Web Working Symposium, Stanford University, 2001.

[Lassila 2002] Ora Lassila: "Taking the RDF Model Theory Out for a Spin", First International Semantic Web Conference (ISWC 2002), Sardinia (Italy), June 2002.

[Patel-Schneider 2003] Peter F. Patel-Schneider, Pat Hayes and Ian Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C working Draft, 2003.

[Riazanov, 2003] A.Riazanov, Implementing an Efficient Theorem Prover, PhD thesis, University of Manchester, 2003.

[Sagonas, 1994] Kostantinos Sagonas, Terrance Swift, and David S. Warren: XSB as an efficient deductive database engine, In ACM Conference on Management of Data (SIGMOD), 1994.

[Yang, 2000] Guizhen Yang and Michael Kifer. FLORA: Implementing an efficient DOOD system using a tabling logic engine. Proceedings of Computational Logic --- CL-2000, number 1861 in LNAI, pp 1078--1093. Springer, July 2000.

[Zou, 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan, TAGA: Trading Agent Competition in Agentcities, Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth International Joint Conference on Artificial Intelligence, Monday, 11 August, 2003, Acapulco MX.

[Zou, 2004] Youyong Zou, Agent-Based Services for the Semantic Web, Ph.D. Dissertation, University of Maryland, Baltimore County, August, 2004.

Papers produced

This is a list of papers that were partially supported by the project.

2005

Li Ding, Tim Finin, Anupam Joshi, Yun Peng, Rong Pan and Pavan Reddivari, Search on the Semantic Web, IEEE Computer, October 2005. http://ebiquity.umbc.edu/paper/html/id/257/

Mark Burstein, Christoph Bussler, Tim Finin, Michael Huhns, Massimo Paolucci, Amit Sheth, Stuart Williams and Michal Zaremba, A Semantic Web Services Architecture, IEEE Internet Computing, pp 52-61, v9, n5, September/October 2005.
http://ebiquity.umbc.edu/paper/html/id/243/

Martin O'Connor, Holger Knublauch, Samson Tu, Benjamin N. Grosof, Mike Dean, William Grosso, and Mark Musen, Supporting Rule System Interoperability on the Semantic Web with SWRL". Proc. 4th International Semantic Web Conference (ISWC-2005), pp. 974-986, held Galway, Ireland, Nov. 6-10, 2005. http://ebusiness.mit.edu/bgrosof/paps/swrl-editor-iswc2005.pdf

Sumit Bhansali and Benjamin N. Grosof, Extending the SweetDeal Approach for E-Procurement using SweetRules and RuleML". Proc. International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005), Galway, Ireland, Nov. 6-10, 2005.
http://ebusiness.mit.edu/bgrosof/paps/sweetdeal-procurement+sweetrules-ruleml2005.pdf

Li Ding, Rong Pan, Tim Finin, Anupam Joshi, Yun Peng and Pranam Kolari, Finding and Ranking Knowledge on the Semantic Web, Proceedings of the 4th International Semantic Web Conference, Galway IE, Nov. 2005. http://ebiquity.umbc.edu/paper/html/id/241/

Li Ding, Tim Finin, Anupam Joshi, Yun Peng, Paulo Pinheiro da Silva and Deborah McGuinness, Tracking RDF Graph Provenance using RDF Molecules, poster paper, Proc. 4th Int. Semantic Web Conf., Galway IE, Nov. 2005. http://ebiquity.umbc.edu/paper/html/id/263/

Akshay Java, Tim Finin and Sergei Nirenburg, Integrating Language Understanding Agents Into the Semantic Web, First Int. Symposium on Agents and the Semantic Web, 2005 AAAI Fall Symposium Series, Arlington VA, 4-6 November, 2005.
http://ebiquity.umbc.edu/paper/html/id/261/

Harry Chen, Filip Perich, Tim Finin and Anupam Joshi, The SOUPA Ontology for Pervasive Computing, in Ontologies for Agents: Theory and Experiences, Valentina Tamma, Stephen Cranefield, Tim Finin and Steven Willmott (Eds), Springer Verlag, June 2005.
http://ebiquity.umbc.edu/paper/html/id/238/

Tim Finin, Li Ding, Rong Pan, Anupam Joshi, Pranam Kolari, Akshay Java and Yun Peng, "Swoogle: Searching for knowledge on the Semantic Web", Intelligent Systems Demonstration, Proc. 20th National Conf. on Artificial Intelligence, July 2005.
http://ebiquity.umbc.edu/paper/html/id/223/

Tim Finin, Li Ding, Lina Zhou and Anupam Joshi, Social Networking on the Semantic Web, Learning Organization Journal, v12, n5, pp 418-435, 2005.
http://ebiquity.umbc.edu/paper/html/id/222/

Shashidhara Ganjugunte, "Extending the Non-monotonic Reasoning Infrastructure for the Semantic Web via Well-founded Negation and incremental Support for Courteous Logic Programs, MS thesis, University of Maryland, Baltimore County, August 2005.

Pavan Reddivari, Tim Finin and Anupam Joshi, Policy based Access Control for a RDF Store, Proc. Policy Management for the Web workshop (position paper), 14th Int. World Wide Web Conf., pp 78-81, 10 May 2005, Chiba, Japan. http://ebiquity.umbc.edu/paper/html/id/219/

Li Ding, Pranam Kolari, Tim Finin, Anupam Joshi, Yun Peng and Yelena Yesha, On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework, poster paper, AAAI Spring Symposium on AI Technologies for Homeland Security, 21-25 March 2005.
http://ebiquity.umbc.edu/paper/html/id/209/

Lalana Kagal and Tim Finin, Modeling Conversation Policies using Permissions and Obligations, in Developments in Agent Communication, Frank Dignum, Rogier van Eijk, Marc-Philippe Huget (Eds), (Post-proceedings of the AAMAS Workshop on Agent Communication, Springer-Verlag, LNCS, January, 2005.

Li Ding, Lina Zhou, Tim Finin and Anupam Joshi, How the Semantic Web is Being Used:An Analysis of FOAF, Proc. 38th Int. Conf. on System Sciences, Digital Documents Track (The Semantic Web: The Goal of Web Intelligence), Hawaii, January 2005.
http://ebiquity.umbc.edu/paper/html/id/184/

Tim Finin, James Mayfield, Anupam Joshi, R. Scott Cost and Clay Fink, Information Retrieval and the Semantic Web, Proc. 38th Int. Conf. on System Sciences, Digital Documents Track (The Semantic Web: The Goal of Web Intelligence), Hawaii, January 2005.
http://ebiquity.umbc.edu/paper/html/id/183/

2004

Youyong Zou, Tim Finin Harry Chen, F-OWL: an Inference Engine for Semantic Web, in Michael G. Hinchey, James L. Rash, Walter F. Truszkowski and Christopher A. Rouff (editors), Formal Approaches to Agent Based Systems, (Proc. 3rd NASA/IEEE Workshop on Formal Approaches to Agent Based Systems, FAABS III, 16-18 April 2004, Greenbelt MD), Springer Ver-

lag Lecture Notes in Computer Science, v3228, December 2004.
http://ebiquity.umbc.edu/paper/html/id/203/

Aykut Firat, Stuart Madnick, and Benjamin N. Grosof, Contextual Alignment of Ontologies for Semantic Interoperability, Proc. 14th Workshop on Information Technologies and Systems (WITS-2004), pp. 200-205. Held in conjunction with the International Conference on Information Systems (ICIS-2004), Washington, DC, Dec. 11-12, 2004.

Li Ding, Tim Finin and Anupam Joshi, Analyzing Social Networks on the Semantic Web, IEEE Intelligent Systems (Trends and Controversies), v9, n1, Jan/Feb 2005.
http://ebiquity.umbc.edu/paper/html/id/202/

Harry Chen, Tim Finin, Anupam Joshi, Filip Perich, Dipanjan Chakraborty and Lalana Kagal, Intelligent Agents Meet the Semantic Web in Smart Spaces, IEEE Internet Computing, v8, n6, November/December, 2004. http://ebiquity.umbc.edu/paper/html/id/201/

Sumit Bhansali, Benjamin N. Grosof, and Stuart Madnick, Towards Ontological Context Mediation for Semantic Web Database Integration: Translating COIN Ontologies Into OWL, (Poster), Proc. 3rd International Semantic Web Conference (ISWC-2004), Hiroshima, Japan, Nov. 7-11, 2004. http://ebusiness.mit.edu/bgrosof/paps/iswc2004-poster-paper.pdf

Lalana Kagal, Tim Finin and Anupam Joshi, Declarative Policies for Describing Web Service Capabilities and Constraints, W3C Workshop on Constraints and Capabilities for Web Services, October, 2004. http://ebiquity.umbc.edu/paper/html/id/193/

Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Joel Sachs, Vishal Doshi, Pavan Reddivari and Yun Peng, Swoogle: A Search and Metadata Engine for the Semantic Web, Thirteenth ACM Conf. on Information and Knowledge Management (CIKM'04), Washington DC, November 2004. http://ebiquity.umbc.edu/paper/html/id/182/

Tim Finin and Joel Sachs, Will the Semantic Web Change Science?, Science Next Wave, September 2004. http://ebiquity.umbc.edu/paper/html/id/189/

Benjamin N. Grosof and Terrence C. Poon, SweetDeal: Representing Agent Contracts With Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions", International Journal of Electronic Commerce (IJEC), 8(4):61-98, Summer 2004.
http://ebusiness.mit.edu/bgrosof/paps/ijec-sweetdeal-exceptions-111903.pdf

Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin and Katia Sycara, Authorization and Privacy for Semantic Web Services, IEEE Intelligent Systems (Special issue on Semantic Web Services, pp 50-56, v19, n4 (July/August), 2004.
http://ebiquity.umbc.edu/paper/html/id/174/

Harry Chen, Filip Perich, Anupam Joshi and Tim Finin, SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, Processing in Mobile Ad-Hoc Networks Proc. Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services, Boston, August 2004.

Lalana Kagal and Tim Finin, A Generic Model for Conversation Specifications and Policies based on Permissions and Obligations, AAMAS 2004 Workshop on Agent Communication (AC2004), July 2004. http://ebiquity.umbc.edu/paper/html/id/166/

Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin and Anupam Josh, Intelligent Agents Meet Semantic Web in a Smart Meeting Room, The Third Int. Joint Conf. on Autonomous Agents and Multi Agent Systems, New York, July 19-23 July, 2004. http://ebiquity.umbc.edu/paper/html/id/156/

William Krueger, Jonathan Nilsson, Tim Oates and Tim Finin, Automatically Generated DAML Markup for Semistructured Documents, in Agent Mediated Knowledge Management, Lecture Notes in Artificial Intelligence, Luder van Elst, Virginia Dignum and Andreas Abecker (Eds.), 2004. (Revised version of 2003 workshop paper).

Harry Chen, Tim Finin and Anupam Joshi, Semantic Web in the Context Broker Architecture, IEEE Conf. on Pervasive Computing and Communications (PerCom), Orlando, March, 2004. http://ebiquity.umbc.edu/get/a/publication/76.pdf

Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin and Katia Sycara, Authorization and Privacy for Semantic Web Services, AAAI 2004 Spring Symposium on Semantic Web Services, Stanford, March 2004. http://umbc.edu/~finin/papers/AAAIaowl2004.pdf

Harry Chen, Tim Finin and Anupam Joshi, An ontology for context aware pervasive computing environments, Knowledge Engineering Review - Special Issue on Ontologies for Distributed Systems, Cambridge University Press, 2004. http://umbc.edu/~finin/papers/ker03a.draft.pdf

2003

Deepali Khushraj, Tim Finin and Anupam Joshi, Semantic Tuple Spaces: A Coordination Infrastructure in Mobile Environments, poster paper, Second Int. Semantic Web Conf. (ISWC2003). http://ebiquity.umbc.edu/paper/html/id/76/

Anugeetha Kunjithapatham, Mithun Sheshagiri, Tim Finin, Anupam Joshi and Yun Peng, Personal Agents on the Semantic Web, poster paper, Second Int. Semantic Web Conf. (ISWC2003).

Harry Chen, Tim Finin and Anupam Joshi, Semantic Web in a Pervasive Context-Aware Architecture, Workshop on Artifical Intelligence in Mobile Systems, the Fifth Annual Conf. on Ubiquitous Computing, 12-15 Oct 2003, Seattle. http://umbc.edu/~finin/papers/ubicomp03-ai.pdf

James Mayfield and Tim Finin, Information retrieval on the Semantic Web: Integrating inference and retrieval, SIGIR Workshop on the Semantic Web, Toronto, 1 August 2003. http://umbc.edu/~finin/papers/sigir03.pdf

Grit Denker, Lalana Kagal, Tim Finin, Massimo Paoucci, Katia Sycara, Security for DAML Web Serviced: Annotation and Matchmaking, Second Int. Semantic Web Conf. (ISWC2003), Sanibel Island FL, October 2003. http://umbc.edu/~finin/papers/iswc03a.pdf

Lalana Kagal, Tim Finin and Anupam Joshi, A Policy Based Approach to Security for the Semantic Web, Second Int. Semantic Web Conf. (ISWC2003), Sanibel Island FL, October 2003. http://umbc.edu/~finin/papers/iswc03b.pdf

Li Ding, Lina Zhou, Tim Finin, Trust Based Knowledge Outsourcing for Semantic Web Agents, 2003 IEEE/WIC Int. Conf. on Web Intelligence (WI 2003), October 2003, Beijing. http://umbc.edu/~finin/papers/wi03.pdf

Harry Chen, Tim Finin and Anupam Joshi, Using OWL in a Pervasive Computing Broker, Workshop on Ontologies in Open Agent Systems, AAMAS 2003. http://umbc.edu/~finin/papers/aamas03a.pdf

Youyong Zou, Tim Finin, Li Ding, Harry Chen and Rong Pan, Using Semantic Web technology in Multi-Agent Systems: a case study in the TAGA trading agent environment 5th Int. Conf. On Electronic Commerce: Technologies, Pittsburg, 1-3 October 2003.

Mithun Sheshagiri, Marie desJardins, Tim Finin, A Planner for Composing Service Described in DAML-S, Workshop on Web Services and Agent-based Engineering, AAMAS, July 20003. http://umbc.edu/~finin/papers/aamas03b.pdf

Harry Chen, Tim Finin and Anupam Joshi, An ontology for a context aware pervasive computing environment, IJCAI workshop on ontologies and distributed systems, IJCAI'03, August, 2003, Acapulco MX. http://umbc.edu/~finin/papers/ijcai03-ont.pdf

Youyong Zou, Tim Finin, Li Ding, Harry Chen and Rong Pan, TAGA: Trading Agent Competition in Agentcities, Workshop on Trading Agent Design and Analysis, held in conjunction with the Eighteenth Int. Joint Conf. on Artificial Intelligence, 11 August, 2003, Acuulco MX. http://umbc.edu/~finin/papers/tagaijcaidraft.pdf

Mithun Sheshagiri, Marie desJardins, Tim Finin, A Planner for Composing Service Described in DAML-S, Workshop on Planning for Web Services, Int. Conf. on Automated Planning and Scheduling, Trento, July 2003. http://umbc.edu/~finin/papers/icaps03.pdf

Lalana Kagal, Tim Finin and Anupam Joshi, A Policy Language for Pervasive Systems, Fourth IEEE Int. Workshop on Policies for Distributed Systems and Networks, Lake Como, 4-6 June, 2003. http://umbc.edu/~finin/papers/policy03.pdf

William Krueger, Jonathan Nilsson, Tim Oates and Tim Finin, Automatically Generated DAML Markup for Semistructured Documents, AAAI Spring Symposium 2003 on Agent-Mediated Knowledge Management (AMKM), 24-26 March, 2003, Palo Alto, California. http://umbc.edu/~finin/papers/AAAISS03.pdf

2002

Tim Finin and Anupam Joshi, Agents, Trust and Information Access on the Semantic Web, SIGMOD Record, v31, n4, December 2002. http://www.acm.org/sigmod/record/issues/0212/SPECIAL/5.Finin.pdf

Subhash Kumar, Anugeetha Kunjithapatham, Mithun Sheshagiri, Tim Finin, Anupam Joshi, Yun Peng and R. Scott Cost, A personal agent application for the semantic web, AAAI Fall Symposium on Personalized Agents, North Falmouth, Nov 15-17, 2002. http://umbc.edu/~finin/papers/aaaiss03.pdf

Urvi Shah, Tim Finin Anupam Joshi, R. Scott Cost and James Mayfield, Information Retrieval on the Semantic Web, 10th Int. Conf. on Information and Knowledge Management, November 2002. http://umbc.edu/~finin/papers/cikm02/cikm02.pdf

William Krueger, Jonathan Nilsson, Tim Oates and Tim Finin, Automatically Generated DAML Markup for Semistructured Documents, National Science Foundation Workshop on Next Generation Data Mining, Baltimore, November 2002. http://umbc.edu/~finin/papers/ngdm02/ngdm02.pdf

Lalana Kagal, Tim Finin and Anupam Joshi, Developing Secure Agent Systems Using Delegation Based Trust Management, Second Int. Workshop on Security of Mobile Multiagent Systems (SEMAS-2002), Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, pp. 27-34, Bologna, 15-19 July 2002. http://umbc.edu/~finin/papers/semas02/semas02/semas02.pdf

Sushama Prasad, Yun Peng and Tim Finin, Using Explicit Information To Map Between Two Ontologies, Workshop on Ontologies in Agent Systems, First Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, Bologna, 15-19 July 2002.

Benjamin Grosof, Mahesh Gandhe and Tim Finin, SweetJess: Translating DamlRuleML to Jess, Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web, First Int. Semantic Web Conf. (ISWC2002), 14 June 2002, Sardinia. http://umbc.edu/~finin/papers/iscw02/

Sushama Prasad, Yun Peng and Timothy Finin, A tool for mapping between two ontologies using explicit information, AAAI-02 Workshop on Meaning Negotiation, July 28, 2002, Edmonton, Alberta, Canada.

R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff, Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, Sovrin Tolia, ' ITTALKS: A Case Study in the Semantic Web and DAML+OIL', IEEE Intelligent Systems, v17 n1, January/February 2002, pp 40-47. http://umbc.edu/~finin/papers/ieeeis02/ .

2001

R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Filip Perich, Charles Nicholas, Harry Chen, Lalana Kagal, Youyong Zou and Sovrin Tolia, ITTALKS: A Case Student in how the Semantic Web Helps, Proc. First Int. Semantic Web Workshop - Infrastructure and Applications for the Semantic Web, pp. 477-494, July, 2001, Stanford. http://umbc.edu/~finin/papers/swws01/

Filip Perich, R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Harry Chen, Lalana Kagal, Youyong Zou and Sovrin Tolia, ITTALKS: An Application of Agents in the Semantic Web, Workshop on Engineering Societies in the Agents' World, 7 July 2001, Prague. http://umbc.edu/~finin/papers/esaw01/

LIST OF ACRONYMS

ACL	Agent Communication Language
DAML-S	DARPA Agent Markup Language for Services
DL	Description Logic
F-LOGIC	Frame Logic
F-OWL	Frame Logic OWL
FIPA	Foundation for Intelligent Physical Agents
FOAF	Friend of a Friend
FOL	First Order Logic
HTML	Hypertext Markup Language
JADE	Java Agent Developement Environnent
JESS	Java Expert System Shell
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
LP	Logic Programming
MYSQL	MySQL system
NAF	Negation as Failure
OIL	Ontology Interchange Language
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
RDF	Resource Description Framework
RDFS	Resource Description Framework for Services
RFP	Request for Proposals
RSS	Rich Site Summary
RULEML	Rule Markup Language
SCLP	Situated Courteous Logic Programming
SHIQ	a family of abstract description logic languages
SHIQ(D)	a family of abstract description logic languages
SQL	Structured Query Language
SWD	Semantic Web Document
SWO	Semantic Web Ontology
TAC	Trading Agent Competition
TAGA	Trading Agents Game in Agentcities
URI	Uniform Resource Identifier
URL	Uniform Resource Locater
WML	Wireless Markup Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSB	a logic programming language